פרולוג
שפת
בינה מלאכותית

ד"ר בלה צירולניקוב



ערך
יצחק עמיהוד

ספרי לימוד והכשרה במדעי המחשב
הוצאת הוד־עמי לספרי מחשבים

פרולוג
שפת
בינה מלאכותית

תודה ליונתן קורפל ואדי סער ממכללת ״קדם״ על עזרתם.

עריכה: יצחק עמיהוד
עריכה לשונית: גילי דיין–פרל
הכנה והבאה לדפוס: זהר עמיהוד

עיבוד תמלילים: WordMill – אינטרסופט בע״מ
הרצת תכניות: TI-Business PRO באדיבות המחלקה לבינה מלאכותית, טקסל אלקטרוניקה בע״מ
IBM-PC

סדר צילום: אל־אות בע״מ
הדפסת לייזר: Canon LBP–II באדיבות קרט בע״מ
לוחות: סיט שרותי אופסט ☎ 266073
דפוס: חידקל בע״מ

תמונות השער, צ׳רלי ו-PS/2 באדיבות יבמ ישראל בע״מ

פרולוג
שפת
בינה מלאכותית

ד"ר בלה צירולניקוב

ערך יצחק עמיהוד

ספרי לימוד והכשרה במדעי המחשב
הוצאת הוד־עמי לספרי מחשבים

PROLOG
The Language of Artificial Intelligence

by

Dr. Bella Tsirulnikov

edited by I. Amihud

הוצאת הוד־עמי
לספרי מחשבים בע"מ

ת.ד. 560, רמת־גן 52105

טל. המשרד: 052-541207

הודפס בישראל
שבט תשמ"ח, ינואר 1988

Hod-Ami Ltd.
P.O.B. 560 Ramat-Gan 52105
ISRAEL, January 1988

על הספר

חקר הבינה המלאכותית התחיל עוד לפני המצאת המחשב האלקטרוני. המחשב היה הכלי לקידום מחקר זה. בשנים האחרונות המחקר בבינה מלאכותית צבר תנופה עם התקנת היישומים המעשיים הראשוניים. נזכיר כמה משטחי היישום הקיימים, או הנמצאים בשלבי מחקר מתקדמים: מערכות שאילתות בשפה הטבעית, מערכות קולטות דיבור, מערכות ראייה, רובוטים, ומערכות מומחה ברפואה, בתעשייה ובניהול. אלו הן מקצת הדוגמאות להיקף הפעילות בתחום ולחשיבותו.

עדות נוספת לחשיבות שמייחסים לנושא הבינה המלאכותית הינה התכנית רחבת ההיקף הנתמכת ע״י ממשלת יפן. היא הקימה קונסורציום של חברות גדולות למימון המחקר ומכון ממשלתי מיוחד להכוונתו ולריכוז המאמצים. המטרה שהוגדרה היא יצירת מחשבים מן ״הדור החמישי״ – מחשבים בעלי אינטליגנציה מלאכותית. ממשלות רבות באירופה הלכו בעקבות יפן ויזמו תכניות דומות, אם כי שאפתניות פחות.

כל העוסק בבינה מלאכותית ובמערכות מומחה חייב להיות מצוי בשתי השפות המקובלות ביותר בתחום זה, LISP ו־PROLOG. LISP ימיה כימי העיסוק בבינה ממוחשבת והשימוש בה נרחב ביותר. PROLOG (תיכנות בלוגיקה), בת שנים מספר בלבד, אך כבר רכשה לה אוהדים ומשתמשים רבים מפאת היותה מתאימה במיוחד לרבים מהיישומים שהוזכרו. השימוש בפרולוג טבעי ונוח ביותר. התכנית היפנית למחשבי הדור החמישי אף בחרה בפרולוג כשפת מחשבים אלה.

על כן, יש לברך על הופעתו של ספר פרולוג ראשון בעברית, פרי עטה של ד״ר ב. צירולניקוב ובעריכתו של י. עמיהוד.

המחברת עשתה עבודה יסודית בריכוז החומר ובהצגתו בצורה מובנית ונוחה ללימוד. היא מתחילה בהדרכה לדרך חשיבה לוגית ודנה במשמעות של תיכנות לוגי, עוברת להדרכת התיכנות בשפת פרולוג בסביבה של פרולוג מסוימת ומסיימת בהנחייה לפיתוח מערכות מומחה בתחומים שונים.

לקורא ניתנת הנחייה והכוונה, אשר יאפשרו לו להמשיך את שלבי הלימוד וההכשרה באמצעות ספרי ההדרכה של גירסאות הפרולוג המסופקות ע״י היצרנים. התיאור של סביבות עבודה דוגמת TURBO-PROLOG, WISDOM-PROLOG ואחרות מכוון למטרה זו.

המחברת מלווה את ההסברים בדוגמאות קלסיות מן הספרות ובדוגמאות הלקוחות מחיי היום יום כמו למשל, ניתוח דבריהם של נואמים, תוכנה לימודית, מערכת מומחה לזיהוי בעלי־חיים, ועוד. בכל פרק ניתנים תרשימים, תרגילים פתורים מלווים בהסברים, הדפסי מחשב (trace) ותרגילים נוספים לחזרה ושינון.

ספר לימוד זה ממלא חלל גדול בחומר כתוב עברית בתחום הנושאים של בינה מלאכותית. בכך אין ספק שהוא יתרום להעלאת רמת המודעות והידע בנושאים אלה, ההופכים במהירות לשטחי לימוד ועבודה מרכזיים בנושאי המיחשוב ומערכות המידע.

פרופ׳ פיליפ עין־דור
אוניברסיטת תל־אביב

ללוי ונעמי

הבעת תודה

המחברת מודה ל:
ד"ר אהוד שפירא, מכון ויצמן למדע;
ד"ר יהודית גל־עזר, האוניברסיטה הפתוחה;
המחלקה לבינה מלאכותית, טקסל אלקטרוניקה בע"מ.

תוכן העניינים

הקדמה

היבטים שונים של שימוש בפרולוג

פרולוג היא שפת תכנות מתקדמת וחדשנית. על היבטים שונים של השימוש בה נשותח בהקדמה זו. בשיחתנו נעסוק בנושאים הבאים:

א. שפת פרולוג מיישמת רעיון חדש של תכנות – **התכנות בלוגיקה**. תכנית של פרולוג הינה אוסף של נוסחאות לוגיות. את הרצת התכנית ניתן לפרש כתהליך של הסקת מסקנות על-פי כללים לוגיים.

ב. פרולוג הינה **שפה הצהרתית**. התכנית של פרולוג מתארת בעיה. באופן מופשט ניתן לומר, כי התכנית של פרולוג מספקת למחשב ידע, שעל פיו מתקבלות המסקנות.

ג. בהתאם למסקנות נעשה עדכון של הידע, מתקבלות מסקנות חדשות וחוזר חלילה. בדרך זו ניתן לתכנן תהליכים מורכבים ומסועפים, אשר הבנתם אינה מושלמת. תהליכים אלה הינם נושא של מחקר בתחום של בינה מלאכותית. בשל כך מכנים את פרולוג בשם **"שפה של בינה מלאכותית"**.

ד. בעת פתרון בעיה, מיישם המנגנון של פרולוג את אחת האסטרטגיות של בינה מלאכותית – **החיפוש לעומק**.

ה. שפת הפרולוג הינה כלי לפיתוח **תוכנה של בינה מלאכותית**, ובכלל זה של מערכות מומחה.

ו. שפת הפרולוג הינה כלי רב-עוצמה לפיתוח **אב-טיפוס** של מערכת תוכנה.

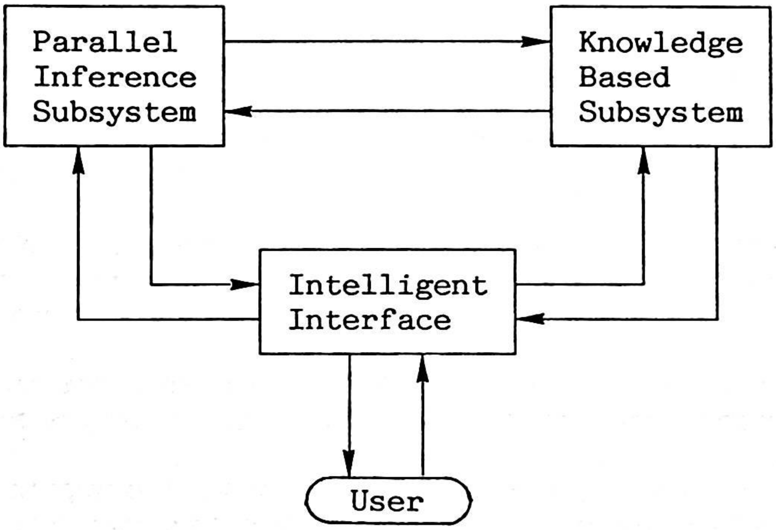
שפת הפרולוג משלבת בתוכה פשטות של שפה הצהרתית, הפועלת על פי כללים לוגיים ברורים, עם תחכום המאפשר ניתוח ותכנות של תהליכים מורכבים ומסועפים, ובשל כך – רבים ימצאו בה ענין.

א. פרולוג – PROLOG - PROgramming in LOGic

בשנות השבעים עסקה קבוצת מדענים מאוניברסיטת מרסיי שבצרפת במחקר בתחום העיבוד של שפות טבעיות. במהלך מחקר זה פותחה שפת תכנות, אשר הציגה מודל חדש של תכנות: **תכנות בלוגיקה**, או **תכנות לוגי**. השפה הכילה מערכת היסק לוגי. באמצעותה ניתן היה להסיק מסקנות על בסיס מידע שהוצג למחשב באמצעות נוסחאות לוגיות. הסקת המסקנות נעשתה על פי כללי היסק לוגיים, ונראתה כחיקוי של תהליך חשיבה אנושית [Colmerauer, 1985].

השפה החדשה כונתה בשם **פרולוג**. השם PROLOG נגזר מצירוף המילים: .PROgramming in LOGic

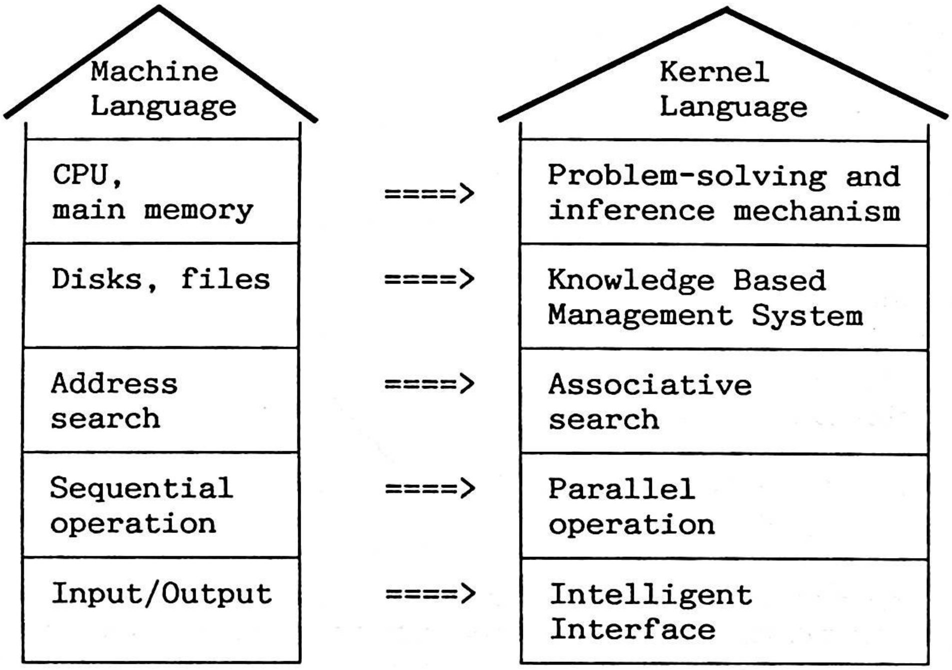
בראשית שנות השמונים התפרסם הפרוייקט היפני של מחשבי הדור החמישי. פרוייקט זה חשף, לדברי היפנים, תפישה חדשה של מבנה המחשב.



איור א' – המחשב של הדור החמישי

לכל מחשב יש שפה הנקראת **שפת מכונה**, אשר תואמת את מבנהו ואת אופן פעולתו. על בסיס שפה זו נבנות גם שפות עיליות. המחשב הנוכחי, הקרוי **מכונת פון-נוימן** (Von Neumann Machine) בנוי על העקרון של חיפוש לפי כתובות. החיפוש מתבצע באופן סדרתי, כתובת אחר כתובת, על כן באופן מופשט נאמר כי תכנית בשפת מכונה היא סדרה של כתובות.
היפנים החליטו לבנות מחשב, אשר יהיה בנוי על העקרון של חשיבה לוגית, ואשר פעולתו העיקרית תהיה הסקה על פי כללי היסק לוגיים [Fuchi, 1984]. הבעיה שתוצג למחשב תבחן בכמה כיוונים במקביל, ומכאן כינויו: **המחשב המקבילי** (Parallel Computer). המחשב יהיה קשור למערכת הידברותית (Interactive) של מאגרי ידע (Knowledge Based Subsystems).
כדי להינתק מן המושגים הישנים ולהדגיש את חדשנות הגישה, כינו היפנים את שפת המכונה של המחשב המקבילי בשם **שפת גרעין** (Kernel Language). הם בנו אותה על בסיס שפת הפרולוג [Kawanobe, 1984]. אחד הנימוקים לבחירת פרולוג כשפת גרעין של המחשב המקבילי היה הנימוק הבא: לוגיקת היחסים הינה כלי להצגת ידע [Nilsson, 1980]. יעודו של המחשב החדש יהיה ייצור ועיבוד ידע, ולכן שפת הפרולוג, שבנויה על בסיס לוגיקת היחסים, מתאימה למטרה זו.

(בהקשר לנאמר לעיל נציין את המאמר [Shapiro, 1986], אשר דן בהתאמה בין שפת תכנות לבין מבנה המחשב).



איור ב' – תפישות חדשות של הדור החמישי

ב. פרולוג – שפה הצהרתית

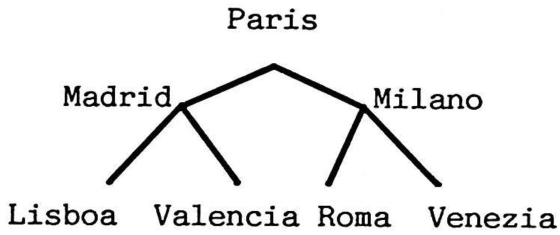
פרולוג הינה שפה סימבולית, בעלת מבנה נתונים דמוי עץ. התכנית של פרולוג מורכבת מעובדות, חוקים ושאילתות. העובדות מתארות את הנתונים, והחוקים מתארים את הקשרים הלוגיים בין הנתונים. יחד הם מתארים את הבעיה. השאילתה, אשר המבנה התחבירי שלה זהה לזה של העובדה, מתקבלת כהוראה לבצע את החישוב ולמצוא את הפתרון. תהליך החישוב נסתר מעיני המשתמש וניתן להציגו כתהליך הסקת מסקנות על בסיס עובדות וחוקים. התהליך מתנהל על פי כללי היסק לוגיים שהינם חלק אינטגרלי של השפה.

שפת הפרולוג פותרת בעיה על פי התאור הכללי שלה, וללא הוראות ביצוע מפורטות. במילים אחרות, צריך להגיד לפרולוג **מה** לעשות. צריך לתאר את הבעיה ולספק את המידע הרלוונטי, אך אין צורך לפרט את צעדי החישוב ולהסביר **איך** נבצע את המשימה. סוג תכנות זה מכונה בשם **הצהרתי**, או **דקלרטיבי** (Declarative). לעומתו התכנות המסורתי, הדורש פירוט מדוקדק של צעדי החישוב (כמו התכנות בפורטרן או בבייסיק), מכונה בשם **פרוצדורלי** (Procedural).

נציג דוגמה של תכנית הצהרתית בפרולוג. התכנית אשר להלן, איננה זקוקה לתיעוד, היא מדברת בעד עצמה. את העובדות והחוקים כותבים ב-text-editor ולאחר הטענתם אל מערכת הפרולוג מציגים שאילתות ומקבלים תשובות.

```
road(paris, madrid).
road(paris, milano).
road(madrid, lisboa).
road(madrid, valencia).
road(milano, roma).
road(milano, venezia).

connected(X, Y) if road(X, Y).
connected(X,Y) if road(X, Link)
           and road(Link, Y).
?- connected(paris, Z).
Z = madrid
Z = lisboa
Z = valencia
Z = milano
Z = roma
Z = venezia
```



הביטויים שמתחילים במילה road הינם עובדות ואלה שמתחילים במילה connected הינם חוקים. נציין, כי אין חשיבות לסדר כתיבתם. האותיות X, Z, Y, הינם משתנים, אשר משמעותם – עיר כלשהי. תיארנו את הכבישים, הסברנו באמצעות החוקים מה פירוש הדבר "עיר קשורה לעיר", הצגנו שאילתה – וזכינו לתשובה. לנושאי הקשר הבין-עירוני עוד נחזור במהלך הספר.

נציין תכונה נוספת של תכנות לוגי, אשר הוצגה בתכנות שלעיל: תכנות לוגי הינו **תכנות מבני** (Structured Programming).

נביא דוגמה נוספת. את בעיית מיון הרשימה נפתור בפרק 9 באמצעות תכנית הצהרתית "מיון טבעי". לשם כתיבתה נתאר את המשימה ואת צעדי הביצוע מבלי לפרטם. להלן התאור. בפרק 9 נתרגם אותו לשפת פרולוג ובכך נסיים את כתיבת התכנית.

* כדי למיין את הרשימה List, צריך לבחור מבין תמורותיה את הרשימה הממויינת Sorted.

* כדי למצוא את התמורות של הרשימה List, שיש בה פריט אחד לפחות, נוציא ממנה פריט אחד X, נחשב את התמורה של שאר הרשימה, ונחזיר את הפריט X.

* רשימה בת פריט אחד הינה ממויינת. רשימה שבה יותר מפריט אחד היא ממויינת, אם שני פריטיה הראשונים מסודרים בהתאם לסדר שנקבע, ושאר פריטיה ממויינים.

בתאור זה לא פרטנו, **איך** נבחר את התמורה הממויינת, **איך** נחשב את התמורה של שאר הרשימה, **ואיך** נבדוק האם שאר פריטיה ממויינים. את הפרטים האלה שפת הפרולוג משלימה בעצמה.

על כן, ניתן להיעזר בפרולוג במקרים בהם אין אנו יודעים באופן ברור וחד-משמעי **איך** לפתור את הבעיה הנתונה. כמובן, שבמקרים כאלו לא ניתן לעצב את התכנית בשיטות הנדסת התוכנה

המקובלות (כמו למשל, DFD). אנשים שמפתחים תוכנה לבעיות אשר אותן הם מבינים בצורה חלקית בלבד, עובדים לפי שיטת **התכנות הבוחן** (Exploratory Programming). תוך כדי כתיבת התכנית הם בוחנים את הבעיה ומחפשים את העיצוב הנכון. שפת הפרולוג תומכת בסוג זה של תכנות.

אף על פי ששפת הפרולוג יודעת במקרים רבים לפתור את הבעיה הנתונה מבלי שהמתכנת יפרט את צעדי החישוב, לא ניתן להתעלם לחלוטין מן הצד הפרוצדורלי. מתכנת שמכיר היטב את מנגנון השפה, יכול לייעל את תהליך החישוב ובכך לחסוך משאבים רבים.

ג. פרולוג – שפה לתכנות של תהליכים מסועפים

הצגת ידע הינה נושא מרכזי בתחומים רבים, המאוחדים תחת השם **בינה מלאכותית** (Artificial Intelligence). בהתאם לכך, שפת פרולוג, אשר משמשת להצגת ידע, הינה **השפה של בינה מלאכותית**.

על אף השימוש הנפוץ במונח "בינה מלאכותית", אין הסכמה כללית לגבי משמעותו. לרוב, משתמשים במונח זה כשמדובר בביצועי המחשב, אשר ניתן לתארם במושגים של תפישה חושית (perception), הסקה (reasoning), אסטרטגיות של חיפוש (search strategies) וכדומה. מושגים אלה הינם נחלת החשיבה האנושית, ומכאן מקורו של המונח "בינה מלאכותית".

בלשון חופשית ניתן לומר, כי בינה מלאכותית הינה תחום, החוקר את מנגנוני התבונה. בחקר זה משתתפים מתמטיקאים, אנשי מחשב, פסיכולוגים, אנשי רוח ועוד. בהקשר למדעי המחשב, בינה מלאכותית עוסקת בעקרונות, שיטות וטכניקות, אשר באמצעותם אפשר לחקות במחשב את תהליך של חשיבה אנושית. שפת הפרולוג מיישמת כמה מהשיטות והטכניקות של בינה מלאכותית בתחומים רבים, ובפרט בתחומים האלה:

* עיבוד והצגת ידע.
* עיבוד שפות טבעיות.
* מערכות מומחה.
* רובוטיקה.

בתחומי היישום של בינה מלאכותית לרוב אי אפשר לתאר את הפתרון באופן חד-משמעי, באמצעות שרשרת צעדים הידועים מראש והמובילים אל הפתרון. יש לכך סיבות שונות. הסיבה העיקרית היא, שעקב הסתעפויות וריבוי אפשרויות, יש יותר משרשרת אחת של צעדים (כלומר הפתרון אינו דטרמיניסטי). ישנה גם סיבה נוספת – לעיתים אין אנו יודעים איך לפרט את הצעדים.

בתחומים אלה אנו מסוגלים במקרים רבים לתאר **מה** לעשות, ואיננו מסוגלים לציין **איך** לעשות. שפת הפרולוג פותרת עבורנו את ה"איך" באמצעות כללי היסק לוגיים שהינם חלק אינטגרלי של השפה. פירוש הדבר הוא, שבמקרים רבים שפת הפרולוג מסוגלת למצוא את הפתרון על פי תאור הבעיה והצגת השאילתה בלבד, מבלי שנציין את פרטי הביצוע. אנו ננצל תכונה זו במקרים בהם אי אפשר לתאר את הפתרון באופן חד-משמעי, עקב ריבוי האפשרויות, או מכיון שהבעיות והתהליכים אינם מובנים באופן מושלם. את התכנית לצורך זה מפתחים בשיטת התכנות הבוחן [Sheil, 1987]: מתארים את הבעיה ומתכנתים אותה בפרולוג, מריצים, מעדכנים, ממשיכים בתאור, מתכנתים וכן הלאה. בסיום מקבלים תכנית הצהרתית, אשר אותה ניתן לייעל במידת הצורך.

ד. אסטרטגיה של חיפוש פתרון

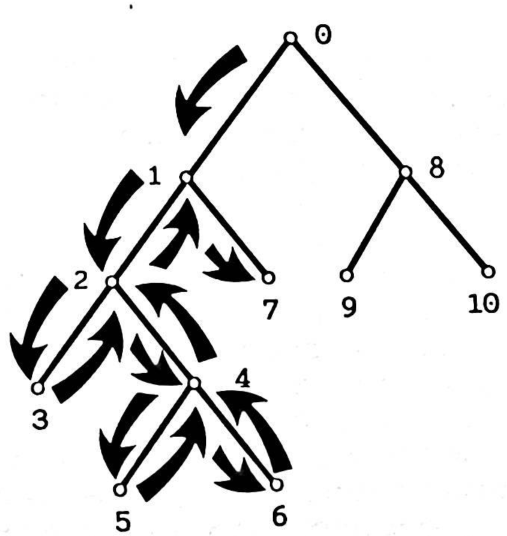
ביישומים של בינה מלאכותית מקובל לתאר את הבעיה באמצעות **מרחב המצבים** (state space). ישנו המצב המצוי – המצב בו נמצאת המערכת כרגע, וישנו המצב הרצוי, שאליו רוצים להגיע. ישנם מצבים נוספים, שאליהם ניתן להגיע מן המצב המצוי. חיפוש הפתרון מתבטא בבניית שרשרת של מצבים, המקשרת את המצב המצוי עם הרצוי. במילים אחרות, על מנת לפתור את הבעיה, יש למצוא דרך להגיע למצב הרצוי. נציין כי במקרים רבים יש יותר מדרך אחת, שמובילה אל הפתרון.

נביא דוגמה של תאור בעיה, אשר ניתן לפרשו באמצעות מרחב המצבים. בפרק 9 פתרנו את בעית המיון באמצעות תכנית "מיון טבעי". המצב המצוי, או המצב ההתחלתי, תואר באמצעות הרשימה הנתונה List. המצב הרצוי תואר באמצעות הרשימה הממויינת Sorted. בתור מרחב המצבים שימשו **תמורות** (permutations) של הרשימה List, אשר אחת מהן היתה הרשימה הממויינת Sorted. המעבר ממצב למצב התבצע באמצעות הפרוצדורה permutation. פרוצדורה אחרת, sorted, בדקה אם המצב החדש הינו מטרת החיפוש, כלומר, הרשימה Sorted.

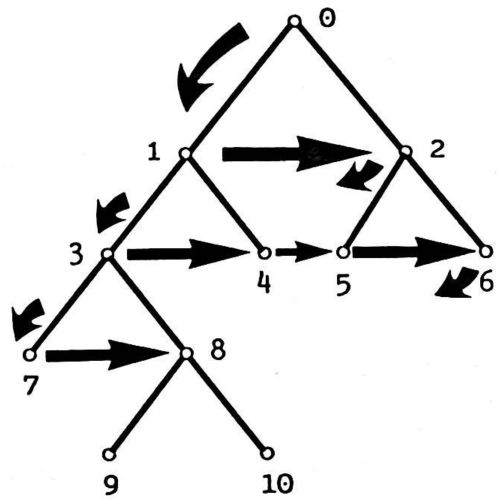
את מרחב המצבים ניתן להשוות למפה, אשר היישובים המסומנים בה הם המצבים, והכבישים הינם אמצעי המעבר ממצב למצב. היישוב בו אנו נמצאים הינו המצב המצוי, או המצב הנתון. היישוב שאליו רוצים להגיע, הוא המצב הרצוי, או המטרה. חיפוש הפתרון כמוהו כחיפוש קשר בין היישובים. כדאי להזכיר, כי בחיפוש זה מחכות לנו הפתעות לא נעימות, כמו לולאות או מבוי סתום. על כל אלו יש להתגבר בעת החיפוש.

ישנם יישובים, שניתן להגיע אליהם באמצעות מערכת הכבישים וישנם יישובים, שניתן להגיע אליהם בשבילי עפר בלבד. מכיון שלא ניתן להגיע אליהם באמצעים המקובלים עלינו, הם כלל אינם רלוונטיים לנושא החיפוש. אוסף המצבים, אליהם ניתן להגיע מהמצב ההתחלתי, מכונה **מרחב החיפוש** (search space). מרחב החיפוש הינו תת-מרחב של מרחב המצבים.

את מרחב המצבים מתארים באמצעות **גרף** (graph). נסמן את המצבים בעיגולים או נקודות ונעביר **קשתות** (arcs) בין המצבים הקשורים אחד למשנהו. התאור שהתקבל הינו **גרף**. המצב בגרף מכונה **צומת** (node). בסוג של גרף, הנקרא **עץ** (tree), נדון בפרק 8.



איור ג' – חיפוש לעומק

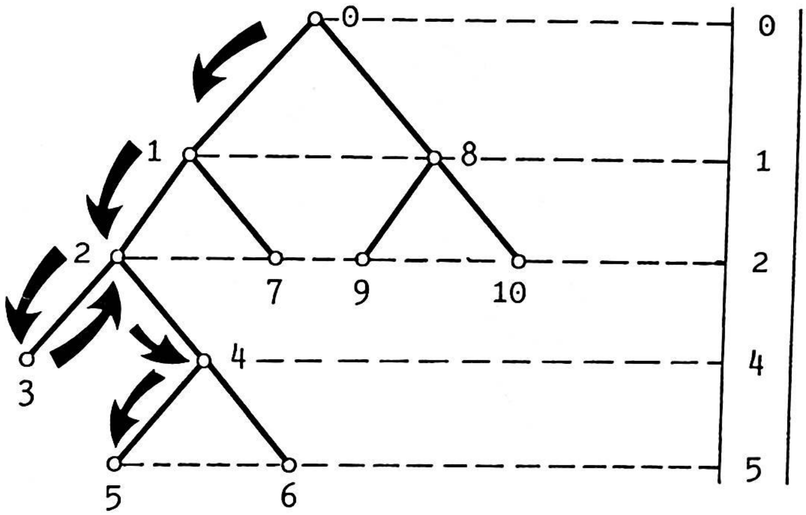


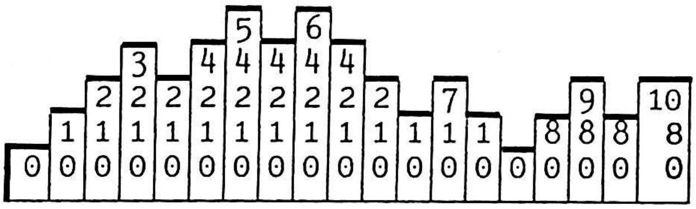
איור ד' – חיפוש לרוחב

קיימות שתי אסטרטגיות בסיסיות של חיפוש הפתרון: האחת – **חיפוש לעומק** (depth-first search), מתוארת באיור ג'; השניה – **חיפוש לרוחב** (breadth-first search), מתוארת באיור ד'. בחיפוש לעומק או לרוחב, מרחב החיפוש נסרק ביסודיות עד אשר נמצא הפתרון. קיימות אסטרטגיות אחרות, אשר בהן מרחב החיפוש נסרק באופן סלקטיבי, בהתאם לקריטריונים שונים, למשל על פי המקדם, המבטא את חשיבותו של הצומת או על-פי קירבתו אל המטרה. חיפוש מסוג זה נקרא **חיפוש האוריסטי** (heuristic search). אם כן, חיפוש האוריסטי הינו חיפוש לעומק או לרוחב בתוספת של פונקציית הערכה. בחיפוש מסוג זה משתמשים כאשר מרחב החיפוש גדול מדי. במקרה זה מדלגים על צמתים בעלי סיכוי נמוך ועל ידי כך מקצרים את זמן החיפוש.

הצמתים שנסרקו על ידי חיפוש לרוחב, מיוצגים באמצעות תור ומתנהלים בשיטת FIFO (First In First Out). לעומת זאת, בחיפוש לעומק משתמשים במחסנית ועובדים בשיטת LIFO (Last In First Out). המנגנון של פרולוג מכיל אלגוריתם של חיפוש לעומק. במקרים בהם החיפוש נקלע למבוי סתום, המנגנון מבצע **נסיגה לאחור** (backtracking).
בסעיפים 7 ו-9 של חטיבה רביעית נביא סדרה של תכניות, אשר בכל אחת מהן נדגיש את האופן שבו אסטרטגיית החיפוש לעומק מיושמת על ידי פרולוג.

נציין, כי בנוסף לחיפוש לעומק, ניתן בפרולוג ליישם גם אסטרטגיות חיפוש אחרות. דבר זה דורש קידוד של האלגוריתמים המתאימים, בשעה שהחיפוש לעומק הינו חלק אינטגרלי של כל תכנית.





איור ה׳ – עקומה של מחסנית בעת חיפוש לעומק

ה. פרולוג – שפה לפיתוח תוכנה של בינה מלאכותית

כפי שאמרנו, המונח "בינה מלאכותית" נועד לתאר את ביצועי המחשב, המחקים תהליך חשיבה אנושית. אחד מחלוצי התחום של בינה מלאכותית, פרופ׳ מרווין מינסקי (Marvin Minsky), תיאר בינה כמושג, שמשתמשים בו כדי לציין פעולה שיכלית, אשר אותה מעריכים, אבל לא מבינים.

הגדרתו של מינסקי משקפת את האופן שבו אנחנו רואים את תהליך העבודה השיכלית. ניקח למשל, את עבודת הרופא. היא מסווגת כנבונה ואינטליגנטית על ידי אלה העומדים מהצד. לעומת זאת, עמיתיו רואים בה שימוש נכון בשיטות וטכניקות, ותו לא. באשר לעבודת המחשב, על פי גישה זו של מינסקי, כשמבינים היטב את תהליך קבלת הפתרון, אין זו בינה מלאכותית, אלא אוסף של עקרונות, שיטות וטכניקות ותו לא. העובדה, שתהליך קבלת הפתרון דומה מאוד לתהליך חשיבה אנושית, אינה רלוונטית כלל וכלל. על פי גישה זו מותר להשתמש במונח "בינה מלאכותית" רק כאשר מדובר בתהליכים, אשר אינם מובנים באופן מלא. עם השיטות שבאמצעותם ניתן לטפל בתהליכים כאלה, נמנים תכנות בוחן ותכנות בלוגיקה. לאחר שהתהליך הזה תוכנת והורץ במחשב, הוא מאבד את זכותו להיקרא בשם "בינה מלאכותית".

בדרך כלל השימוש במונח "בינה מלאכותית" חופשי בהרבה מזה הנגזר עליו על פי ההגדרה שלעיל. ישנו אוסף של מוצרים, אשר תהליך פעולתם מובן היטב, ובכל זאת מכנים אותם בשם

המוצרים (או התוכנה) של בינה מלאכותית. דוגמה מסוג זה היא מערכת מומחה המכילה ידע של מומחה ואסטרטגיות, שעל פיהן המומחה מפעיל אותה.

מערכת מומחה מכילה מנגנון הסקה, אשר לרוב אינו תלוי בידע של המומחה, ובשל כך הוא משמש כלי לבניית מערכות נוספות. בחטיבה הרביעית נציג מערכת מומחה לימודית (Toy Expert System). כמו-כן, נציג קונכיה (Shell) של מערכת מומחה, המפעילה מפרש-על (metainterpreter) של פרולוג בתור מנגנון הסקה.

שיטת התכנות הבוחן הינה השיטה המקובלת בפיתות תוכנה של בינה מלאכותית. שיטה זו מיישמת את גישתו של מינסקי, משמע, באמצעותה ניתן לפתח תוכנה בנושאים אשר הבנתם איננה מושלמת.

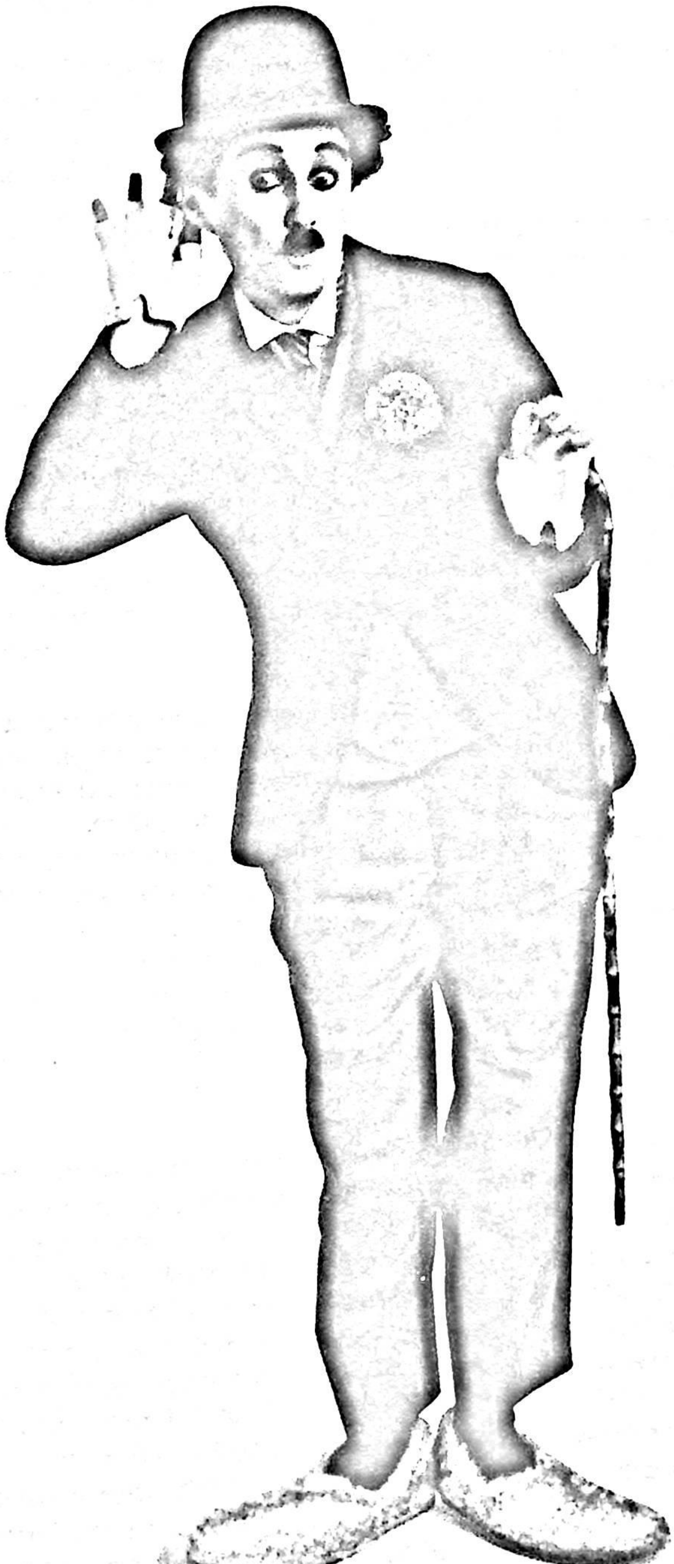
ו. פרולוג – שפה לבניית אב-טיפוס של מערכת תוכנה

לדעת מומחים, השלב הקשה ביותר בבניית מערכות תוכנה הינו שלב האיפיונים, אשר בו מפתחים את תפישת המערכת. בשלב זה מחליטים מה המערכת מבצעת ואיך היא תיבנה. המשתמש, אשר אינו בקיא בתהליכי פיתוח תוכנה, חייב להגדיר את דרישותיו כך, שעל פיהן ניתן יהיה לפתח מערכת. בדומה, מהנדס תוכנה, אשר אינו בקיא בתחום עיסוקו של המשתמש, חייב להבין ולספק את דרישותיו. מכאן שהמשתמש אינו יודע מה הוא רוצה, והמפתח אינו יודע מה הוא בונה.

פרופ׳ פרדריק ברוקס, אחד מהמפתחים של משפחת מחשבי IBM-System 360, ממליץ במאמר שכתב [Brooks, 1987] להתחיל את פיתוח המערכת מבניית אב-טיפוס (prototype). באמצעות אב-טיפוס מיישמים את תפישת המערכת בקווים כלליים ומאפשרים לשני הצדדים, למשתמש ולמפתח, להכיר וללמוד את הדרישות של הצד השני. המשתמש מציג את דרישותיו בצורה נאותה, והמפתח מציע לו פתרונות מתאימים. נציין כי לשיטת פיתוח תוכנה, אשר מתחילה מבניית אב-טיפוס, ישנם תומכים רבים [Barnes, 1987].
אב-טיפוס של מערכת יש לבנות מהר, בזול, ותוך שיתוף פעולה מקסימלי עם המשתמש. פרולוג הינה שפה הצהרתית, בעלת מבנה נתונים גמיש. היא מצוידת במנגנוני הסקה וחיפוש ותומכת בתכנות בוחן ובעיצוב מובנה מלמעלה למטה (top-down structured design). בשל תכונות אלו, פרולוג הינה כלי רב-עוצמה לפיתות מהיר של אב-טיפוס למערכת תוכנה.

* * *

את הנאמר בהקדמה נסכם כך. פרולוג הינה שפה הצהרתית, אשר מכילה את האלגוריתם של חיפוש לעומק עם נסיגה מבוקרת לאחור. בשל כך היא מאפשרת להכין תכניות, שאינן מפרטות את צעדי הביצוע בזה אחר זה. תכניות אלו מכילות מידע, הרלוונטי לפתרון הבעיה. המידע מוצג בצורת עובדות, המתארות את הנתונים, וחוקים המתארים את הקשרים הלוגיים בין הנתונים. ניתן לכתוב תכנית בפרולוג, כאשר התהליך מתואר באופן כללי, אבל פרטיו אינם ברורים. את התכניות של פרולוג ניתן לעצב בשיטת התכנות הבוחן, ובשל כך השפה נוחה במיוחד לעיצוב מהיר של אבות-טיפוס, לבניית מערכות מומחה, לתכנות תהליכים מסועפים עם אפשרויות ביצוע רבות, וכדומה.
נסיים ונאמר: פרולוג הינה שפה המיישמת היום את הטכנולוגיה של העתיד, ועל-כן רבים ימצאו בה עניין. מחד גיסא, המתחילים יראו בה שפה הצהרתית, הפועלת על-פי כללים לוגיים ברורים, אשר נוח להתחיל ממנה את ההיכרות עם עולם המחשב; מאידך גיסא, אנשי מחשב מנוסים יהיו מעוניינים להכיר כלי חשוב בעל עוצמה רבה, שנועד ליישומים מתוחכמים ולמחשבי הדור החמישי.



הנחיות למשתמש

מדריך ללומד – שיטות ויעדי הלימוד

מדריך למתכנת מקצועי – סביבות עבודה

מדריך ללומד – שיטות ויעדי הלימוד

ספר זה נכתב בעקבות הקורסים האלה:
– עקרונות של בינה מלאכותית ושפת פרולוג (60 שעות), לאנשי מחשב.
– שפת פרולוג ומערכות מומחה (60 שעות), לאנשי מחשב.
– תכנות לוגי ושפת פרולוג (48 שעות), לקהל רחב.
– חוג לפרולוג במסגרת פרוייקט שנתי של מכון וייצמן, לתלמידי כיתה י'.

הספר מתחיל במבוא "מבט חטוף על פרולוג" שנכתב ברמה אינטואיטיבית פשוטה ומהווה סקירה של נושאים, אשר ידונו בהמשך. הודגש הבדל בין תכנות הצהרתי (declarative) לבין תכנות פרוצדורלי (procedural) (ראה פרק 2 בספר [Martin, 1985]).

החטיבה הראשונה של הספר דנה בנושאים של לוגיקה. שפת לוגיקה פסוקית מוצגת כהפשטה (abstraction) של שפה טבעית (עברית או אנגלית), המאפשרת תאור פורמלי של תהליך הסקת מסקנות.

החטיבה השניה דנה במבנה של תכנית לוגית. הושם דגש על הגישה ההצהרתית (ראה חלק 2 בספר [Yazdani, 1984]). על-פי גישה זו תכנית לוגית מתוארת כמכשיר **להצגת ידע** (knowledge representation). נציין, כי בנוסף להוראת תכנות לוגי, גישה זו נפוצה בהוראת תוכנה הידברותית (interactive), כדוגמת "גיליון עבודה אלקטרוני" (electronic worksheet). **נציין גם** כי תוך עיון בחטיבה זו הקורא מתוודע לעבודה עם בסיסי נתונים (ניסוח של יחסים, הצגת שאילתות וכדומה).

החטיבות השלישית והרביעית דנות בשיטות וטכניקות של תכנות מעשי בפרולוג. הושם דגש על ניסוח בעיה תוך שימוש בפרוצדורות ובמבני נתונים מורכבים. שיטת הלימוד מבוססת על הגישה האפיסטמולוגית (epistemological), המקובלת בהוראת בינה מלאכותית (ראה פרק 7 בספר [Papert, 1980]). גישה זו כוללת פירוק הבעיה למרכיביה הפשוטים, ניתוח של קשריהם הלוגיים ותאורם בשפה פורמלית. אף-על-פי שתאור המרכיבים נעשה בפרולוג, גישה זו אינה תלויה בשפה מסוימת, אשר על כן השיטות הנלמדות (כדוגמת מרחב המצבים, או generate-and-test), ישמשו את הקורא בתחומים נוספים. יצוין כי נעשה שימוש באלמנטים של הגישה האלגוריתמית, מבלי להכנס לנושאים תיאורטיים כגון סיבוכיות (complexity), נכונות (correctness) וכדומה (הנושא של סיבוכיות תכנית לוגית נדון במאמר [Shapiro, 1984]; נושאים אחרים נידונים בספר [Sterling & Shapiro, 1986]). נציין, כי המשימות הלימודיות של החטיבה הרביעית (אשר דנה בכתיבת תכניות יישום) מדורגות בהתאם להתקדמות של הקורא בחומר הנלמד.

הספר מכיל חומר רב לתרגול. חלק מהתרגילים ניתן לצורך הדגמה תוך הגשת הנושאים. חלק אחר מוגש במרוכז ומהווה מטלה לימודית בעלת תוכן מוגדר.

הספר מיועד לקהל מגוון של קוראים. אנשי מחשב, שהתחנכו על מושגים של תכנות פרוצדורלי וגישה אלרגוריתמית, וודאי ימצאו אתגר בהיכרות עם דרך חשיבה אחרת. לקוראים, שעבורם פרולוג היא שפת התכנות הראשונה, נמליץ להתחיל את הלימודים מפרק 5, לאחר מכן לקרוא את המבוא, לעבור לפרק הראשון ולהמשיך לפי הסדר העולה של הפרקים.

הספר מתאים ללימוד עצמי וגם ללימוד בכיתה. התכניות והתרגילים הוכנו והורצו בסביבה של WISDOM-PROLOG (המתוארת בנספח א').

יעדי הלימוד:

מבוא
– להכיר מושגים בסיסיים של תכנות לוגי:
* ייצוג ידע באמצעות עובדה, חוק, שאילתה.
* עיבוד ידע וחישוב שאילתה, כתהליך של הסקת מסקנות לוגיות.
* חישוב שאילתה על ידי מערכת הפרולוג באמצעות התאמה, הצבה ונסיגה.

פרק 1
– ללמד ייצוג של ידע בשפה פורמלית באמצעות:
* פסוק אטומי.
* קשרים לוגיים.
* מושגים של אמת (T) ושקר (F).

פרק 2
– להכיר את הבסיס הלוגי של המושגים:
* שקול (שווה-ערך).
* נובע (גורר).

פרק 3
– להכיר את המושג "עקביות".

פרק 4
– לבנות מערכת היסק.
– ללמד תהליך הסקה לוגית.

פרק 5
– להציג תכנית לוגית כבסיס נתונים.
– ללמד מושגי יסוד של בניה ושימוש בבסיסי נתונים:
* יחס (relation).
* עצם (object).
* עובדה (fact).
* משתנה (variable).
* שאילתה (query).

פרק 6
– לייצג ידע באמצעות מושגים בסיסיים של לוגיקת היחסים:
* פרדיקט (predicate) – מתאר את היחס.
* ארגומנט (argument) –מתאר את העצם.
* כמת (quantifier) – מציין האם העובדה המוצגת נכונה עבור **עצם אחד לפחות**, או עבור **כל עצם**.
* נוסחה (well-formed formula) – מתארת את הקשרים הלוגיים בין היחסים.

פרק 7
– ללמד ייצוג של ידע באמצעות:
* כללים "אם-אז" (IF-THEN).
* חוקים רקורסיביים.

פרק 8
– להכיר את מבנה הנתונים של פרולוג.
– לדעת לנסח בעיה תוך שימוש במבני נתונים מורכבים.

פרק 9
– להעמיק את הידע במבנים רקורסיביים.
– לדעת לקרוא תכנית לוגית.
– לדעת לעצב תכנית לוגית.

פרק 10
- להכיר את תהליך האיחוד (unification).
- לבנות מודל חישובי של תכנית לוגית ולישמו בפרולוג.

פרק 11
- לדעת לשנות זרימה של תכנית לוגית.
- לדעת לייעל את תהליך חישוב השאילתה.

פרק 12
- ללמד להשתמש באמצעי עזר שונים, המסופקים על ידי מערכת הפרולוג (פרדיקטי המערכת).

חטיבה רביעית
- לכתוב תכניות הידברותיות (interactive) בעלות תכנים לימודיים.
- להכיר אסטרטגיות של פתרון בעיה באמצעות תכנות לוגי.
- לדעת לבנות מערכת מומחה.
- לדעת לפתח מערכת מומחה באמצעות הקונכיה (shell).
- להכיר את המפרש-על (metainterpreter) של פרולוג.

מדריך למתכנת מקצועי – סביבות עבודה

מקובל כי הוראת פרולוג מתנהלת בשני שלבים. נתאר אותם בקצרה.

שלב ראשון – רמה לוגית

בשלב זה בונים תשתית:
- ⋆ מתוודעים לרעיון המונח ביסוד השפה: הסקה לוגית באמצעות מחשב.
- ⋆ לומדים תכנות הצהרתי שבמהלכו אומרים למחשב **מה** לעשות, והמחשב יחליט **איך** לעשות.

לשלב זה הקדשנו את החטיבות הראשונה והשניה.

שלב שני – רמת הביצוע

בשלב זה לומדים תכנות מעשי, ובכלל זה:
- ⋆ מבנה הנתונים של השפה.
- ⋆ מודל חישובי של תכנית לוגית.
- ⋆ שיטות וטכניקות של עיצוב התכנית.

לשלב זה הקדשנו את החטיבות השלישית והרביעית.

מתכנת מקצועי שסיים קורס לפרולוג ומכיר את השפה ברמה הלוגית וברמת הביצוע, עובר לשלב הבא:

שלב שלישי – רמת הסביבה

בשלב זה נעשית התקנה של תוכנת פרולוג, והמתכנת לומד להכיר את סביבת העבודה ואיפיוניה, לדוגמה:
- ⋆ הפרדיקטים של המערכת (evaluable predicates).
- ⋆ מישקי תקשורת (communication interfaces, links).
- ⋆ אמצעי ניפוי (debugging tools, trace).

ועוד.

המתכנת "נחשף" לאלמנטים של הסביבה כבר בשלב השני של הלימוד. הדוגמאות והתכניות שבספר זה נבנו בסביבת עבודה של WISDOM-PROLOG, אשר מתוארות בפרוט בנספח א'. כדי להקנות לקורא תחושה של עבודה מקצועית תחת WISDOM, הצגנו מספר תכניות: עיבוד מערכים (תכנית א'), בניית העורך (תכנית ב') ועוד. נספח ג' מתאר את TI-PROLOG המיושם במכונת LISP, ונספח ה' מתאר את גירסת טורבו-פרולוג. נציין, כי חלק מהתכניות בספר זה אינן מתאימות לטורבו-פרולוג בשל המבנה המיוחד שלו. נספח ב' מתאר את האופרטורים של פרולוג ומציג את רשימת האופרטורים המוגדרים מראש של WISDOM-PROLOG.

המתכנת לומד את סביבת העבודה בעזרת מדריך למשתמש של יצרן התוכנה. מדריך זה, בדרך כלל, אינו מהווה ספר לימוד של פרולוג ולכן אינו דן בשלבים הראשון והשני. היצרן מצפה שהמשתמש מכיר את העקרונות של תכנות לוגי, והמדריך מנחה אותו איך להשתמש בגירסת הפרולוג שבידיו.

לעתים קורה כי בשל אילוצים שונים, המתכנת חשוף לשלב השלישי מבלי לעבור את השלבים הראשון והשני. כלומר, המתכנת לומד פרולוג באמצעות מדריך (user manual), אשר אינו מתאים להיות ספר לימוד. כתוצאה, נוצר אצלו רושם מוטעה, כי פרולוג הינה שפה פרוצדורלית נוספת, שכן המדריך מתאר עבודה מעשית במחשב סדרתי ואינו מדגיש את הרעיון ודרך חשיבה של תכנות לוגי.

פרולוג שונה בתכלית משפות פרוצדורליות כגון פסקל, פורטרן וקובול. בנספח ד' השווינו בין append פרוצדורלי לבין append לוגי, וגם הסבנו את תשומת הלב למאמרים הדנים בנושא זה.

כדי לנצל את עוצמתה של פרולוג, נמליץ להתחיל את כתיבת תכניות היישום בדרך הבאה:

- בשלב ראשון נכתוב מודולים הצהרתיים, אשר בהם יושם דגש על הקשרים הלוגיים (רמה לוגית).
- בשלב שני נעבד מודולים אלה באמצעות ארגון פיסקאות, Cut וכדומה, כדי לייעל אותם (רמת ביצוע).

בדרך זו, בשלב הראשון נתרכז בחשיבה לוגית, ובשלב השני נתחשב עם אפשרויות של ביצוע סדרתי. תוך כדי עבודה מעשית בפרולוג המתכנת יפסיק "לעבוד בשלבים" ויתחיל "לחשוב פרולוג".

מבוא

מבט חטוף על פרולוג

לא נגזים אם נאמר, כי אין בימינו אדם בן-תרבות, שיכול להתעלם מקיומו של המחשב.

המחשב הוא מכונה. מאז ומתמיד בנו אנשים מכונות, ש"הגבירו" את יכולתם והגדילו את שליטתם בטבע ובסביבה. הם בנו מכוניות כדי "לתגבר" את הרגליים, והמציאו טלפון כדי להתגבר על בעיות מרחק.

אף על פי שהמחשב הוא מכונה, הוא שונה באופן מהותי ממכונות אחרות. כנגד מכונה, המבצעת פעולה אחת, או לכל היותר כמה פעולות, המחשב מסוגל לבצע תפקידים רבגוניים, אשר לרוב אין ביניהם כל קשר. מחשב אחד יכול לשריין מקומות לטיסה ולנווט מטוסים וטילים. הוא יכול לנתב תקשורת ארוכת טווח ובו-זמנית להכין חשבון טלפון.

המחשב הוא מכונה מיוחדת. הוא נבנה כדי לתגבר את מוחו ותבונתו של האדם.

1. תכנות לוגי כאמצעי לעיבוד וייצור ידע

המחשבים הראשונים, אשר נבנו בשנות החמישים, התקבלו כמגבירי כושר החישוב, ובעזרתם מהירות החישוב גברה בהרבה על זו של האדם. במשך השנים גדלו המהירות, הזכרון, האמינות ועם זאת – גדל גם התיחכום. מכלי חישוב המחשב הפך להיות כלי לעיבוד נתונים ולהפקת מידע, החיוני לקבלת החלטות חשובות. על-כן, מכלי חישוב הפך המחשב לכלי לייצור מידע.

התהליך של ייצור ידע על בסיס מידע נעשה תחילה בידי האדם. על מנת ליצור ידע, היה דרוש לנתח את הקשרים הלוגיים בין פרטי המידע ולהסיק מסקנות בדרך לוגית. היום, תהליך עיבוד וייצור ידע נעשה ע"י המחשב. תהליך זה כולל פיתוח כלים שונים, החל משפות תכנות, אשר באמצעותן ניתן לייצג ידע במחשב, וחלה במערכות תומכות החלטה ומערכות מומחה [Luconi & others, 1986].



איור מ-1 – שפת פרולוג הינה שפה לעיבוד ידע

מהו ידע? לא נתיימר לענות על שאלה זו. בין יתר הדברים, ניתן גם לומר כי ידע הוא סוג משאב, אשר המחשב הינו כלי לשמירתו ולמיחזורו. המחשב הינו כלי המאפשר ליצור ידע חדש על בסיס הידע הקיים.

הצגת ידע למחשב הוא אחד הנושאים המרכזיים של המחקר בתחום של בינה מלאכותית. למונח "בינה מלאכותית" (Artificial Intelligence) יש תומכים רבים מחד, ומתנגדים רבים מאידך. מבלי להיכנס לויכוח בנושא זה, נביא כמה מהגדרותיו. הגדרה אחת אומרת, כי "בינה מלאכותית" הינה אומנות, תורה וטכנולוגיה של בניית מערכות מחשב, אשר מחקות פעילות שיכלית של בני-אדם. הגדרה אחרת טוענת, כי "בינה מלאכותית" הינה תחום של מדעי המחשב, העוסק בשיטות להצגת ידע במחשב והסקת מסקנות באמצעותו. הגדרה נוספת גורסת כי "בינה מלאכותית" הינה חקר של פעילות שיכלית של בני-אדם באמצעות מודלים חישוביים.
לכל ההגדרות יש נקודה משותפת: הצגת ידע במחשב.

פרולוג היא שפה להצגת ידע. בעזרתה מיישמים סוג תכנות חדש – **תכנות לוגי**, אשר שונה מהתכנות הפרוצדורלי המקובל.
תהליך התכנות מתחיל בניסוח המשימה ומסתיים בכתיבת תכנית מחשב. בתכנות פרוצדורלי התכנית שמתקבלת בסיומו היא סדרת הוראות, המפרטות את צעדי ביצוע של המשימה. ההוראות מאורגנות בהתאם לסדר ביצוען, אשר בדרך כלל אינו מבטא את דרך החשיבה הלוגית, שהובילה אל הפתרון. בעת עיון בתכנית, עוסקים לרוב בפרטים הטכניים של תפעול התכנית, ופחות – במהלך הלוגי שלה.
שיטת התכנות הפרוצדורלי מחייבת תיעוד מסודר ומפורט. בהכנתו נתקלים לעתים בבעיות, שכן מתכנתים רבים, שרגילים לחשוב במונחים של שפת תכנות פרוצדורלית, מתקשים להביע את מחשבותיהם ורעיונותיהם בשפה אחרת.

בתכנות לוגי המצב שונה. בצורה מופשטת ניתן לומר כך: מספיק לנסח את המשימה ולתאר את צעדי הביצוע. אין צורך לפרטם. במילים אחרות, צריך לספק למחשב ידע שעל פיו הוא יסיק את המסקנות. ניתן, אם כך, לומר, כי תאור המשימה הינו תכנית והינו גם תיעוד.

נראה את תהליך תכנות לוגי באמצעות דוגמה פשוטה.
הנהלת קופת חולים הזמינה מערכת לייעוץ בנושא הרגלי אכילה נכונים. ההמלצות מיועדות

לקבוצות שונות של האוכלוסייה: ילדים, מבוגרים, קשישים, נשים בהריון וכן הלאה. ההמלצה לילדים למשל, אומרת כי עליהם לאכול מזון, שמכיל מרכיבים הנחוצים כדי לגדול. במילים אחרות, ילד אוכל סוג מזון מסויים, אם הסוג הזה מכיל מרכיב הנחוץ כדי לגדול. בצורה פורמלית נבטא את ההמלצה הזאת כך:

```
child eats food IF food contains ingredient
                   AND ingredient is needed for growth.
```

ברוב הגירסאות של פרולוג נהוג לסמן את מילת הקשר IF באמצעות הסימן ":-", ואת מילת הקשר AND באמצעות פסיק (","). האוכל עליו יש להמליץ איננו ידוע, כלומר הוא הנעלם, או במילים אחרות, המשתנה הלוגי. נסמן אותו במילה המתחילה באות אנגלית גדולה: Food. מילה זאת מבטאת את משמעות המשתנה.

לפנינו חלק של מערכת הייעוץ. תכנות לוגי הינו תכנות מבני, אשר על כן חלק זה של המערכת הינו תכנית לוגית בזכות עצמו.

```
eats(child,Food):-
     contains(Food, Ingredient),
     needed_for(Ingredient, growth).

eats(adult,Food):-
     contains(Food, Ingredient),
     needed_for(Ingredient, energy).

contains(meat, protein).
contains(milk, calcium).
contains(vegetables, vitamins).
contains(fruits, vitamins).
contains(meat, fats).

needed_for(calcium, growth).
needed_for(protein, growth).
needed_for(vitamins, growth).
needed_for(vitamins, energy).
needed_for(fats, energy).
```

תכנית לוגית מ-1 – המלצת מזון

את שתי הפיסקאות הראשונות, המתחילות במילה eats, ניתן לכנות בשם **כללים** או **חוקים** (rules) לאכילה נכונה. הפיסקאות, שמתחילות במילים needed_for, contains, הן **עובדות** (facts) הכוללות מידע על מרכיבי המזון. לסדר כתיבתן של הפיסקאות אין כל חשיבות, ולכן כתבנו אותן בסדר שביטא את תהליך החשיבה הלוגית.

התכנית פשוטה וברורה. המשימה מתבטאת באמצעות המילה eats, שמתארת את היחס בין קבוצת האוכלוסיה, כגון ילדים או מבוגרים, לבין המזון Food. המילים: needed_for, contains, מבטאות את היחס בין המזון Food לבין המרכיב Ingredient. שתי הפיסקאות הראשונות

מתארות באמצעות המילים IF (הסימן ":-") ו-AND (הסימן ",") את הקשרים הלוגיים בין היחסים needed_for, contains, eats, הפיסקאות האלה מספקות את הידע הדרוש, שעל פיו המחשב יוכל להסיק את המסקנות.

המשימה, דהיינו, המלצת מזון, מתבצעת באמצעות הצגת שאילתה למחשב. בתגובה, המחשב מציג את ההמלצה. תהליך קבלת התשובה נסתר מהמשתמש. אמנם, יש אפשרות שהמשתמש, אם ירצה בכך, יעקוב אחר תהליך חישוב התשובה באמצעות ההוראה "עקוב-trace" (ראה סעיף 7.6). אם המשתמש רוצה לקבל הסברים ונימוקים להמלצות שהתקבלו, ניתן לעצב את מערכת הייעוץ בצורה של מערכת מומחה (Expert System). מבנה גמיש של נתונים מאפשר "להלביש" על תכנית של פרולוג מנגנון מיוחד, המכונה בשם קונכיה (Shell) של מערכת מומחה (בנושא זה נדון בחטיבה האחרונה).

נציג שאילתות ונעיין בתשובות.

שאילתה	?- eats(child,Food).
תשובות	Food = meat; Food = milk; Food = vegetables; Food = fruits.
שאילתה	?- eats(adult, Food).
תשובות	Food = vegetables; Food = fruits; Food = meat.

איור מ-2 – המלצת מזון: שאילתות ותשובות

בהמשך הספר נביא גירסה נוספת של התכנית הזאת (ראה תכנית Food בפרק 7).

את המונחים, כגון: יחס, עצם, משתנה, שבהם נעזרנו בסעיף זה, נלמד ביתר הרחבה בפרק 5. מקורם של מונחים אלו בלוגיקת היחסים, אשר אותה נלמד בפרק 6. את הידע להמלצת המזון הצגנו באמצעות חוקי אכילה נכונה. סקירה של מבנה החוק נביא בסעיף הבא. את נושא החוק נלמד בהרחבה בפרק 7.
לסיכום נאמר, שתהליך של תכנות לוגי קצר ויעיל בהרבה מתהליך תכנות פרוצדורלי. פשטותו ובהירותו גורמות לשיתוף פעולה בין המתכנת לבין המשתמש ומקילות על עבודת הכנת התיעוד למערכת התוכנה.

2. ייצוג ידע באמצעות עובדות וחוקים

המחשבים הראשונים התמחו בחישובים מתמטיים. היתה לכך סיבה טובה: החישובים התבססו על נוסחאות, שניסוחן היה מדויק וחד-משמעי. צריך היה רק לתרגם את הנוסחאות האלה לשפה מדוייקת אחרת: שפת תכנות.

לעומת זה, את הידע אין מבטאים בנוסחאות, אלא בשפה מדוברת, כגון עברית או אנגלית. שפות אלו, המכונות גם **שפות טבעיות**, אינן מדויקות. מבחינה תחבירית הן בנויות על פי כללים, אבל יש בהן יוצאים מן הכלל, שמספרם רב למדי. מבחינה סמנטית יש בהן עירפול ודו-משמעויות, ריבוי גוונים, ביטויים עקיפים ואי דיוקים למיניהם.

מכאן, שעל מנת לעבד וליצור ידע באמצעות המחשב, עלינו לבחור דרך ברורה וחד-משמעית כדי לייצג ידע. במילים אחרות, עלינו לבחור מודל מתאים לייצג ידע. לשם כך ניעזר במונחים של לוגיקת היחסים (אשר אותה נלמד בפרק 6). את הידע נציג באמצעות עובדות וחוקים.

- העובדות ישמשו לייצוג עצמים ולתאור תכונותיהם והיחסים שביניהם.
- החוקים ישמשו להגדרת תכונות ויחסים חדשים על בסיס התכונות והיחסים הקיימים.

את החוק נכתוב באופן הבא: את התכונה שהחוק מגדיר נכתוב בראש; את העצמים, שמהווים בסיס להגדרה, נכתוב בהמשך. את הראש ואת ההמשך נקשור באמצעות מילת-קשר "אם" (IF). הדוגמה שלהלן מתארת את תרגומו של "סיפור קיץ" לעובדות וחוקים של מודל לייצוג ידע.

סיפור קיץ	
בשפה מדוברת	באמצעות חוקים
בקיץ חם.	חם אם קיץ.
כשחם, הולכים לים.	ים אם חם.
בים שוחים ומבלים, ובקיצור - עושים חיים.	שוחים אם ים. מבלים אם ים. עושים_חיים אם ים.

איור מ-3 – ייצוג ידע באמצעות חוקים

כמובן, שאין זו חובה להציג את המשפט המילולי "בקיץ חם", דווקא כחוק "חם אם קיץ". בשל הרבגוניות שלו, ניתן היה להציגו באמצעות חוק אחר, למשל: "קיץ אם חם", ניתן היה גם להציגו כעובדה: "בקיץ_חם". העדפנו את החוק "חם אם קיץ" מטעמי שמירת קשר לוגי פורמלי, אשר משמעותו תוברר כשנבנה שרשרת של מסקנות (ראה איור מ-7).

נמשיך בסיפור הקיץ:

המשך סיפור קיץ	
בשפה מדוברת	באמצעות עובדה
הקיץ בא.	קיץ.

איור מ-4 – ייצוג ידע באמצעות עובדה

גם את המשפט המילולי "הקיץ בא" ניתן היה להציג בצורות שונות. הסיבה, אשר לפיה בחרנו להציגו באמצעות עובדה "קיץ" תובהר בסעיף 4, כשנדון בנושא הסקת מסקנות ובניית שרשרת של הוכחה (ראה איור מ-7).

3. מרכיבים של תכנית לוגית: עובדה, חוק, שאילתה

לפנינו תכנית לוגית, שנכתבה בתעתיק של הפרולוג העברי. הסימן ":-" מסמל את מילת הקשר "אם" (המילה "אם" היא תרגומה של המילה IF, והסימן ":-" הוא תמונת ראי של הסימון המקביל בפרולוג האנגלי).

חם :- קיץ. ים :- חם. שוחים :- ים. מבלים :- ים. עושים_חיים :- ים.	חוקים
קיץ.	עובדה

תכנית לוגית מ-2 – סיפור קיץ

תכנית לוגית מורכבת מעובדות, חוקים ושאילתה. לחלק התכנית שמכיל עובדות וחוקים, קוראים בשם מסד נתונים (Database) או מסד ידע (Knowledge Base). נציין, כי בשם "תכנית" משתמשים גם כשמדובר על אוסף של עובדות וחוקים בלבד, ללא השאילתה. המינוח בתחום התכנות הלוגי עדיין איננו ממוסד, ועל-כן ייתכנו מונחים שונים לשם ייצוג של מושג זהה. על כל פנים, נשתדל לשמור על המינוח הבא: אוסף, אשר מכיל חוקים, נכנה בשם מסד ידע או בסיס ידע. אוסף, המכיל עובדות בלבד נכנה בשם מסד נתונים או בסיס נתונים.

את העובדות והחוקים מחסינים בקובץ, אשר ישרת את מערכת הפרולוג בעת ההרצה. "הרצה של תכנית לוגית" הינה הצגה של שאילתה וקבלת תשובה, אשר מתבססת על העובדות והחוקים. השאילתה היא המרכיב השלישי של התכנית הלוגית.

נציג שאילתות על-פי מסד הידע של "סיפור קיץ" ונעיין בתשובות:

שאילתה	תשובה
קיץ ?	כן
חם ?	כן
ים ?	כן
שוחים ?	כן
משתזפים ?	לא
מבלים ?	כן
שותים_קולה ?	לא
עושים_חיים ?	כן

איור מ-5 – סיפור קיץ: שאילתות ותשובות

תכנית לוגית מיישמים באמצעות שפת התכנות פרולוג. הקו התחתון מיועד לחבר מילים, בדומה למקף המחבר בשפה הטבעית. משתמשים בו בפרולוג העברי והאנגלי כאחד: שותים_קולה, עושים_חיים, needed_for.

4. איך מסיקים מסקנות

את התשובה החיובית לשאילתה "קיץ?", מקבל המחשב מכיון שמסד הידע "סיפור קיץ" מכיל את העובדה "קיץ". את התשובה החיובית לשאילתה "חם?" מקבל המחשב בדרך הסקה לוגית. תהליך זה נלמד בצורה מפורטת בפרקים 7 ו-10; כאן נביא הסבר תמציתי. נעיין בחוק: "חם :-קיץ" (נזכירכם כי ":-" הוא הסימן התחבירי למילת הקשר "אם"). התשובה החיובית לשאילתה "חם?" התקבלה כתוצאה משיקולים אלו: נתון כי הגיע הקיץ (העובדה "קיץ"), ונתון כי בקיץ חם (החוק "חם :-קיץ"). מכאן **נובע** כי אכן חם (במושג "נובע" נדון בפרקים 2 ו-3).

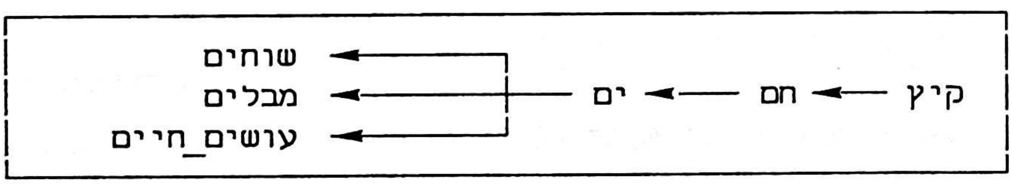
נתונים	קיץ. חם אם קיץ.
מסקנה	חם

איור מ-6 – הסקת מסקנות לוגיות

לנתונים שבאיור מ-6 יש שם נוסף: **אקסיומות**, או הנחות יסוד. גם למסקנה שבאיור זה יש שם נוסף: **משפט**. על כך נרחיב את הדיון בפרק 4.

התשובה השלילית לשאילתה "משתזפים?" מציינת כי למחשב אין ידע שגור (common sense). אם לא נאמר במפורש, באמצעות העובדה, כי אכן משתזפים, ואם אין חוק שעל פיו ניתן להסיק כי משתזפים, פרוש הדבר **שלא** משתזפים. הוא גם הדין לגבי השאילתה "שותים_קולה?".

להלן שרשרת של מסקנות שהוסקו על פי העובדה והחוקים של התכנית הלוגית "סיפור קיץ":



איור מ-7 – שרשרת של מסקנות

בנושא של הסקת מסקנות לוגיות נדון בהרחבה בחטיבה הראשונה של הספר. את היישום של תהליך הסקת מסקנות על ידי מערכת הפרולוג נציג בפרקים 7 ו-10.

את הנאמר לעיל נסכם כך:
– תכנית לוגית היא אמצעי להצגת ידע למחשב ולהסקת מסקנות בדרך לוגית על ידי המחשב.
– תכנית לוגית מורכבת מעובדות, חוקים ושאילתות.

– החוק של התכנית הלוגית בנוי בצורת פסוק, המורכב משני פסוקים באמצעות מילת הקשר "אם" (IF).

על תהליך הבנייה של פסוקים מורכבים באמצעות מילות קשר נדון בפרק 1 במסגרת של לוגיקה פסוקית.

5. התאמה והצבה

נחזור לתכנית "המלצת מזון" ונתאר איך התקבלה התשובה הראשונה לשאילתה:
?-eats(child, Food).

המילה Food הינה משתנה, המסמל את המזון, שעליו יש להמליץ. משמעותו של המשתנה בתכנית לוגית, ובפרט בתכנית פרולוג, שונה ממשמעותו בתכנות הפרוצדורלי. השם "משתנה" בתכנות הפרוצדורלי הוא כינוי נוסף לתא הזכרון. אשר על כן, להוראת בייסיק: X=X+1, יש משמעות של השמה (assignment). במילים אחרות, ההוראה הזאת גורמת למחשב להגדיל ב-1 את התוכן של התא X.

בתכנות לוגי, לביטוי X=X+1 כלל אין משמעות. המשתנה הינו משתנה לוגי, כלומר הוא מסמל עצם שאיננו ידוע, ואחת המטרות של תכנית לוגית היא לגלות את העצם הזה. הגילוי נעשה באמצעות **ההתאמה וההצבה**. כאשר מציגים למערכת הפרולוג את השאילתה, היא מאתרת את החוק המתחיל בביטוי eats(child,Food). האיתור נעשה על ידי בדיקת התאמה של התווים במילה eats של השאילתה, אל המילה eats של החוק. המילה השנייה – child של השאילתה תושווה אל המילה השנייה – child של החוק, וכן הלאה. אחר כך עוברת המערכת לביטוי: contains(Food,Ingredient). היא מאתרת מבין העובדות את העובדה contains(meat,protein), ומציבה את העצם meat תחת המשתנה Food, ואת העצם protein תחת המשתנה Ingredient. ההצבה תחת המשתנים Food ו-Ingredient נעשית בכל אחת מהופעותיהם בתוך החוק.

לאחר ההצבה, החוק נראה כך:

```
eats(child,meat):- contains(meat, protein),
                   needed_for(protein, growth).
```

את המשמעות הלוגית של החוק הזה ניתן לבטא באמצעות המשפט הלשוני הבא:

```
child eats meat if meat contains protein
    and protein is needed for growth.
```

מערכת הפרולוג כבר "יודעת" כי בשר מכיל פרוטאינים, שהרי היא איתרה את העובדה contains(meat, protein). עכשיו היא מאתרת את העובדה needed_for(protein, growth), מאמתת את העובדה, כי הפרוטאין דרוש לשם גידול, ומכאן מסיקה כי ילד אכן אוכל בשר. שלבי האיתור וההצבה מתוארים באיור מ-8:

1	שאילתה	`?- eats(child,Food).`

2	חוק	`eats(child,Food):-`
		`contains(Food,Ingredient),`
		`needed_for(Ingredient,growth).`

3	עובדה	`contains(meat,protein)`
4	הצבה	`Food = meat, Ingredient = protein`
5	עובדה	`needed_for(protein,growth)`
6	תשובה	`Food = meat`

איור מ-8 – שלבי קבלת התשובה

המושגים "התאמה" ו"הצבה" יידונו באופן מפורט בפרק 10. עידון תהליך ההתאמה, המכונה בשם **איחוד**, יידון גם הוא בפרק זה.

את תהליך קבלת התשובה לשאילתה בלשון חופשית ניתן לתאר כחיקוי של תהליך הסקת מסקנות בדרך לוגית על ידי האדם.

6. נסיגה

המחשב, שבנוי על פי הארכיטקטורה של פון-נוימן, מבצע את הפעולות באופן סדרתי, כלומר בזו אחר זו. משתמע מכאן, כי תכנית המחשב מתבצעת בצעדים בדידים, המוגדרים באופן חד-משמעי. אין להתיר מצב, שבו התכנית נמצאת מול אפשרויות רבות, וללא הוראה ברורה כיצד עליה לפעול. במילים אחרות, תכנית שנועדה לביצוע במחשב של פון-נוימן, היא תכנית דטרמיניסטית.

לעומת זה, תהליך הסקת מסקנות בדרך לוגית איננו תהליך דטרמיניסטי. אנו עומדים "במבוכה" מול אפשרויות רבות, מנסים, טועים, נסוגים, מנסים אפשרות אחרת, וחוזר חלילה.

אנו עומדים מול מצב כזה: מחד, תכנית לוגית היא בעקרון תכנית לא דטרמיניסטית (על כך נדון בפרק 10); מאידך, תכנית לוגית מיושמת באמצעות שפת פרולוג במחשב של פון-נוימן, הפועל באופן דטרמיניסטי. לכאורה, נקלענו למבוי סתום.

שפת הפרולוג פותרת את הבעיה באמצעות הדמיית תהליך לא-דטרמיניסטי של חיפוש אחר תשובה. ההדמיה נעשית בשיטות ובטכניקות שונות (ראה למשל, פרק 10). חלק מהשיטות הן נחלת מערכת הפרולוג וחלקן יזומות על ידי המתכנת. אחת השיטות, השייכת למערכת הפרולוג, היא **נסיגה**.

כאשר מערכת הפרולוג מגיעה למבוי סתום בעת חיפוש אחר תשובה, היא מבטלת את הצעד האחרון אשר ביצעה, ובודקת האם קיימת אפשרות להמשיך את החיפוש בדרך אחרת. פעולה זאת נקראת **נסיגה** (backtracking).

נביא דוגמה. בעת חיפוש אחר תשובה לשאילתה ?-eats(adult, Food) פונה פרולוג אל החוק השני, אשר התחלתו תואמת לשאילתה. מן הביטוי eats(adult,Food) פרולוג פונה אל הביטוי contains(Food,Ingredient), ומבצעת את ההצבה:

```
Food = meat,    Ingredient = protein
```

מאחר והמערכת איננה מאמתת את הביטוי needed_for(protein,energy), היא מבצעת **נסיגה:** היא מבטלת את ההצבה אשר ביצעה, ומבצעת הצבה אחרת:

```
Food = milk,    Ingredient = calcium
```

גם את ההצבה הזאת היא מבטלת. ההצבה הבאה:

```
Food = vegetables,  Ingredient = vitamins
```

מובילה אל התשובה:

```
Food = vegetables
```

לסיום נציין, כי ההצבה תחת המשתנה איננה מתבצעת למעשה. כלומר, תוכן כתובות הזכרון איננו משתנה. ההצבה נעשית על ידי ציון קשר בין כתובת המשתנה לבין כתובת העצם שהוצב תחתיו.

1	שאילתה	?- eats(adult,Food).
2	חוק	eats(adult,Food):- contains(Food,Ingredient), needed_for(Ingredient,energy)
3	עובדה	contains(meat,protein)
4	הצבה	Food = meat, Ingredient = protein
5	בדיקה	needed_for(protein,energy) ???
		נסיגה
3	עובדה	contains(milk,calcium)
4	הצבה	Food = milk, Ingredient = calcium
5	בדיקה	needed_for(calcium,energy) ???
		נסיגה
3	עובדה	contains(vegetables, vitamins)
4	הצבה	Food = vegetables, Ingredient = vitamins
5	בדיקה	needed_for(vitamins,energy) ???
6	עובדה	needed_for(vitamins,energy).
7	תשובה	Food = vegetables

איור מ-9 – נסיגה

7. סיכום: סקירת הנושאים אשר יידונו בספר

לסיום נציין את עיקרי הדברים אשר למדנו, ואת הקשרם בהמשך הספר. נסקור גם נושאים אחרים, אשר נלמד בהמשך.

א. פרולוג הינה שפה לייצוג ועיבוד ידע. באמצעותה ניתן להגדיר ולנתח קשרים לוגיים בין עצמים שונים. על קשרים לוגיים נדון במסגרת של לוגיקה פסוקית בפרקים 1, 2, 3.

ב. פרולוג הינה שפה הצהרתית. התכנית של פרולוג מספקת למחשב ידע בנושאים רלוונטיים מבלי לפרט את הצעדים לביצוע המשימה. את תהליך הביצוע ניתן לתאר כתהליך הסקת מסקנות בדרך לוגית. נלמד זאת בפרק 4, במסגרת הדיון בלוגיקה פסוקית.

ג. המידע הרלוונטי מיוצג בתכנית פרולוג באמצעות עובדות. את המידע שואבים באמצעות השאילתות. על העובדות והשאילתות של פרולוג נדון בפרק 5.

ד. מקורו של פרולוג – בלוגיקת היחסים. נושאים של לוגיקת היחסים, הרלוונטיים לפרולוג, יידונו בפרק 6.

ה. את הידע מבטאים בתכנית לוגית, ובפרט בתכנית של פרולוג, באמצעות חוקים. את החוק של פרולוג, מבנהו והשימוש בו נלמד בפרק 7.

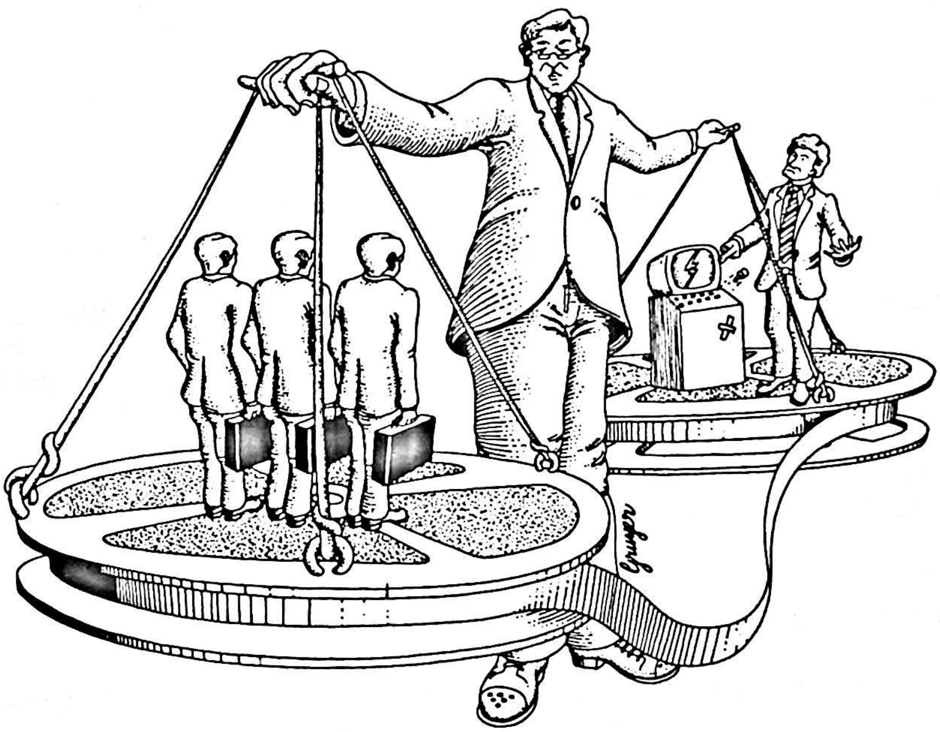
ו. מבנה הנתונים של פרולוג הינו גמיש ועשיר. לא ניתן היה להרחיב את הנושא במסגרת פרק זה. את מבני הנתונים נלמד בפרק 8, שבו נכיר גם את פעולות החשבון שניתן לבצען בפרולוג.

ז. מבנה, המכונה בשם **רשימה**, מאפשר להציג קבוצת עצמים בתור עצם אחד. הרשימה של פרולוג שונה מרשימה של שפה פרוצדורלית בכך, שאין מצהירים על אורך הרשימה וסוג איבריה. הרשימה הינה באורך כלשהו וכוללת בתוכה איברים מסוגים שונים. על מבנה הרשימה ואפשרויות השימוש בה נדון בפרק 9.

ח. בפרק 10 נדון במושגים של הצבה, התאמה ואיחוד, ונבנה מודל חישובי של תכנית לוגית.

ט. בפרקים 11, 12 נדון במרכיב של מערכת הפרולוג, המכונה בשם פרדיקט המערכת, אשר תפקידו לייעל את תהליך התכנות הלוגי.

י. בחטיבה הרביעית נציג אוסף של תכניות יישום בנושאים שונים ונסביר את דרך בנייתן והפעלתן.



חטיבה ראשונה:

מבוא ללוגיקה פסוקית

תכנות בפרולוג הינו תכנות באמצעות לוגיקה. תכנית של פרולוג נבנית על-פי כללים לוגיים, ותהליך הפעלתה מתפרש כתהליך של הסקת מסקנות בדרך לוגית. בחטיבה זו ובפרק 6 של החטיבה הבאה, נדון בתשתית הלוגית של פרולוג.

לוגיקה היא שפה, אשר משתמשים בה לייצוג מידע. מבחינת המבנה היא שפה פורמלית, הבנויה על-פי כללי תחביר ברורים וחד-משמעיים וללא יוצאי דופן. מבחינת עושר הביטוי היא "ישרה כמו סרגל": אין בה רמזים, עירפול וריבויי משמעות, שהם נחלת שפות טבעיות. לכל פיסת מידע, המיוצג באמצעות לוגיקה, יש משמעות של אמת או משמעות של שקר, ורק אחת מהן, וישנם כללים ברורים לקביעת המשמעות הזאת.

בחטיבה הראשונה נדון בנושאים של לוגיקה פסוקית, כלומר לוגיקה, אשר בה המשפט הלשוני על כל חלקיו (נושא, נשוא ומושא) הינו היחידה הקטנה ביותר שאינה ניתנת לחלוקה. במסגרת זו נכיר את המושגים של הוכחה ושל מערכת היסק. את הנושאים אלה נארגן באופן הבא:

פרק 1: פסוק וערך אמת שלו

בפרק זה נעסוק **בתחביר** כלומר, בשאלה איך בונים את ביטויי השפה ממרכיביה השונים. מרכיבי השפה שנלמד בפרק זה הם **פסוק אטומי**, **פסוק מורכב**, **קשרים** (ק' פתוחה, ש' קמוצה).
בפרק זה גם נלמד לקבוע אם פסוק נתון הינו **אמיתי** (true) או **שיקרי** (false). כל פסוק מייצג מידע כלשהו ומכאן, שכדי לקבוע את משמעות המידע עלינו לקבוע את אמיתות הפסוק. הקביעה בדבר אמת או שקר היא קביעה סמנטית. **הסמנטיקה** הינה הקשר בין האלף-בית של השפה לבין מה שנאמר בה. במילים אחרות, היא המשמעות של מה שנאמר בעזרת הסמלים התחביריים של השפה.

פרק 2: יחסים סמנטיים בין שני פסוקים

משמעות דומה ניתן להציג בדרכים שונות. אפשר לומר: "אלך לים", ואפשר גם לומר: "לא יתכן שלא אלך לים". אם נתעלם מגוונים ורמזים המאפיינים את השפה הטבעית, ניתן לראות כי שני הפסוקים האלה מבטאים מידע זהה. הפסוקים האלה הינם **שקולים**, ובכך נדון בפרק זה.

ישנם פסוקים בעלי משמעות קבועה, כמו למשל, הפסוק: "היום יש גשם או אין גשם". פסוק זה מכונה **טאוטולוגיה**, והוא תמיד אמיתי. הפסוק אשר תמיד שיקרי, מכונה בשם **סתירה**.

נכיר גם את היחס הסמנטי בין שני פסוקים, המתבטא באמצעות המושג **נובע**. מושג זה נפוץ בחיי יום-יום. בפרק זה נכיר את הבסיס הלוגי שלו.

פרק 3: יחסים סמנטיים בין פסוק לבין קבוצה של פסוקים

העובדות והחוקים של פרולוג, שמייצגים את הבעיה, הם **קבוצה עקבית** של פסוקים והשאילתה היא פסוק אשר **נובע** ממנה. בפרק זה נדון במושג של קבוצה עקבית ונבחן פסוקים הנובעים מקבוצה זו.

פרק 4: מנגנון ההוכחה של לוגיקה פסוקית

תכנית של פרולוג ניתן להציג כמערכת היסק, ואת חישוב השאילתה ניתן לתאר כתהליך של **הוכחה** על פי **כללי היסק לוגיים**. בפרק זה נדון במושג של היסק. תורת ההיסק עוסקת בהוכחות. ההוכחה היא כלי לאימות הפסוק. במילים אחרות, כדי לדעת כי פסוק הוא אמיתי, עלינו להוכיח אותו.

מתוך כללי ההיסק, אשר נכיר, נקדיש תשומת לב מיוחדת **לכלל הניתוק** (modus ponens), בשל חשיבותו בהקשר לפרולוג.

● ● ●

מקורו של פרולוג בלוגיקת היחסים, שהיא שפה עדינה ועשירה יותר משפה של לוגיקה פסוקית. נעיין בשני הטיעונים האלה:

- כל אדם הוא בין תמותה.
- סוקרטס הוא אדם.

מטיעונים אלה נובעת המסקנה, כי סוקרטס הוא בן-תמותה. השפה של לוגיקה פסוקית, אשר בה הפסוק על כל חלקיו הוצג כיחידה אחת, הינה דלה מכדי לטפל בהסקת מסקנה זו. בפרק 6 בחטיבה השניה, נכיר את לוגיקת היחסים ובאמצעותה נציג חלקים שונים של פסוק.

לסיום נציג את הטיעונים המפורסמים שלעיל בתעתיק של פרולוג:

בן-תמותה(X) :- אדם(X).
אדם(סוקרטס).

?-בן-תמותה(סוקרטס).
כן

נושאי הדיון בלוגיקה נבחרו בהתאם למיקומם וחשיבותם במסגרת של פרולוג. לקוראים, אשר עניינם בלוגיקה חורג ממסגרת זו, נמליץ לפנות לספרי לימוד, אשר כמה מהם הוזכרו ברשימה הביבליוגרפית (למשל [Church,1956]).

פרק 1

פסוק וערך-אמת שלו

בפרק זה נדון במושגים הבאים:

- פסוק אטומי,
- ערך-אמת של פסוק,
- פסוק מורכב,
- קשרים לוגיים,
- לוח-אמת של קשרים.

1.1 ערכי אמת TRUE ו-FALSE

פסוק הוא משפט תחבירי בשפה כשלהי, אשר ניתן ליחס לו משמעות של אמת או משמעות של שקר, ורק אחת מהן.

נעיין בדוגמאות אלו:

- לונדון היא בירת אנגליה (אמת).
- סוקרטס הוא יווני (אמת).
- המספר 2 קטן מן המספר 1 (שקר).

לא כל משפט לשוני הוא פסוק. לא ניתן למשל לייחס את המשמעות של אמת או שקר לשאלה "מה השעה?" או לקריאה "אוי ויי!" גם משפטים כמו: "תן לי את הספר", או "לך לעזאזל" (המכונים ציווים), אינם ראויים לתואר אמת או שקר ולכן אינם זכאים להקרא בשם פסוק.

לנוחותנו ניעזר במונח **ערך אמת של פסוק**. קיימים אם-כן, שני ערכי אמת:

- **אמת** אותו נהוג לסמן ב-T (ראש המילה TRUE שמשמעותה "אמיתי").
- **שקר** אותו נהוג לסמן ב-F (ראש המילה FALSE שמשמעותה "כוזב" או "שקרי").

את הנאמר לעיל נסכם כך: משפט תחבירי הוא פסוק, אם ניתן לייחס לו ערך-אמת אחד ויחיד: T (אמת), או F (שקר).

אין אנו מניחים שערך האמת של כל פסוק ידוע לנו. נתבונן למשל במשפט הלשוני:
האיש הזה גנב אבטיח.

המשפט הזה בהחלט ראוי לתואר אמת או שקר, ורק לאחד מהם, ולכן הוא פסוק. נדון בערך האמת שלו: אם האבטיח לא נגנב, אז ערך האמת של הפסוק הזה הוא F (שקר); אם הוא כן נגנב ואם על פי עדי ראיה האיש הזה הוא הגנב, יהיה ערך האמת של הפסוק הנ"ל T (אמת). ומה הוא ערך האמת של הפסוק אם האבטיח אכן נגנב ואין עדי ראיה? במקרה זה לא נדע את ערך האמת שלו. נסכם ונאמר, כי המשפט הלשוני "האיש הזה גנב אבטיח" הוא פסוק, שהרי יש לו ערך-אמת אחד ויחיד: T (אמת), או F (שקר), אף על פי שהערך הזה אינו ידוע לנו.

תרגיל 1.1

א. התבונן במשפטים הבאים וקבע אילו מהם פסוקים ואילו אינם פסוקים:
1) יורד גשם.
2) מה קרה?
3) בשנת 1642 בנה המתמטיקאי הצרפתי פסקל את מכונת החישוב הראשונה.
4) הסתלק מכאן.
5) המחשב הראשון ENIAC נבנה בשנת 1946.
6) גודלו של ENIAC היה: 30 מטר אורך, 3 מטר גובה, 1 מטר רוחב.
7) קדימה, אל מחשבי הדור החמישי!

ב. התבונן בפסוקים הבאים וקבע על פי מיטב הבנתך, מהו ערך האמת של כל אחד מהם:
1) העיר תל-אביב נוסדה בשנת 1936.
2) הקיסרית ליוויה הרעילה את בעלה אוגוסטוס.
3) אי אפשר ללמוד מבלי להתאמץ.

1.2 פסוק אטומי, פסוק מורכב, קשרים

נעיין בפסוקים הבאים:
– אני נוסע לים.
– אני נוסע לטיול.

בהמשך חטיבה זו נעסוק בפסוקים המורכבים מפסוקים אחרים, כדוגמת פסוקים אלה:
– אני נוסע לים **וגם** נוסע לטיול.
– אני נוסע לים, **או** נוסע לטיול.
– אני **לא** נוסע לים.
– **אם** אני נוסע לים, **אז** אני לא נוסע לטיול.

בהמשך נלמד, כי תכנית לוגית היא אוסף פסוקים הקשורים אחד למשנהו. כדי להבין את השיטות לבניית התכנית ודרכי הטיפול בה, נבנה שפה, אשר בניגוד לשפה הטבעית תהיה חסרת עירפול, רמזים, וריבויי משמעויות. תהיה זו שפה פורמלית, הבנויה על פי כללים מוגדרים ובעלת תחביר חד-משמעי לחלוטין. באמצעותה ניתן יהיה לייצג פסוקים של שפות טבעיות ולהבין את הדרך בה אנחנו קושרים פסוק אחד למשנהו.

פסוק של שפה יכול להיות בנוי מפסוקים "פשוטים" יותר, כדוגמת הפסוק: "אני נוסע לים **וגם** נוסע לטיול", או בניסות אחר: "אני נוסע לים **וגם** לטיול". פסוק כזה הינו **פסוק מורכב**. ישנם פסוקים, אשר לא ניתן לפרקם לפסוקים "פשוטים" יותר מבלי לשנות את משמעותם (נסו למשל

לפרק את הפסוק "אני נוסע לים"). פסוק, אשר לא ניתן לפרקו יקרא **פסוק יסודי** או **אטומי**. המילים שבאמצעותן בונים פסוקים מורכבים: **וגם**, **או**, **לא** וכן הלאה, יכונו **קשרים** (ק' פתוחה ו-ש' קמוצה). ~~להלן רשימת הקשרים שבהם נדון:~~

הכינוי העברי	הסימון העברי	הסימון הלוגי	הסימון האנגלי	הכינוי האנגלי
השלילה	לא	$\neg$	NOT	negation
הגימום	ו, וגם	$\wedge$	AND	conjunction
האיווי	או	$\vee$	OR	disjunction
האימוז	אם-אז	$\rightarrow$	IF-THEN	implication
האימום	אם-ורק-אם	$\leftrightarrow$	IF-AND-ONLY-IF	equivalence

איור 1.1 – קשרים לוגיים

הקשרים AND, OR, IF-THEN, IF-AND-ONLY-IF הם קשרים דו-מקומיים (connectives), שכן הם יוצרים פסוק חדש משני פסוקים נתונים. הקשר NOT, המכונה **קשר השלילה**, הוא קשר חד-מקומי.

בהמשך נלמד לקבוע ערך-אמת של פסוק מורכב, על-פי ערכי-אמת של הפסוקים שמהם הוא מורכב. לצורך זה נסמן את הפסוקים שמהם בנוי פסוק מורכב באותיות לועזיות קטנות: a, b, c,..., p, q. למשל, את הפסוק "אני נוסע לים" נסמן ב-a ואת הפסוק "אני נוסע לטיול" נסמן ב-b. את הפסוקים שלעיל, המורכבים מהפסוקים a ו-b, נסמן כך:

הפסוק בעברית	הפסוק בתעתיק לוגי: עם סימון לוגי	הפסוק בתעתיק לוגי: עם סימון אנגלי
אני נוסע לים וגם לטיול	$a \wedge b$	a AND b
אני נוסע לים, או לטיול	$a \vee b$	a OR b
אני לא נוסע לים	$\neg a$	NOT a
אם אני נוסע לים, אז אני לא נוסע לטיול	$a \rightarrow (\neg b)$	IF a THEN (NOT b)

איור 1.2 – דוגמאות של שימוש בקשרים

נעזר בדוגמה נוספת. נסמן ב-a את הפסוק "השמש זורחת". נסמן ב-b את הפסוק "השקדיה פורחת". נבנה פסוקים, המורכבים מ-a ו-b באמצעות הקשרים.

הפסוק בעברית	הפסוק בתעתיק לוגי	
	עם סימון לוגי	עם סימון אנגלי
השמש זורחת והשקדיה פורחת	$a \wedge b$	a AND b
השמש לא זורחת, וגם השקדיה לא פורחת	$(\neg a) \wedge (\neg b)$	(NOT a) AND (NOT b)
אם השמש זורחת, אז השקדיה פורחת	$a \rightarrow b$	IF a THEN b
אם השקדיה לא פורחת, אז השמש לא זורחת	$(\neg b) \rightarrow (\neg a)$	IF (NOT b) THEN (NOT a)

איור 1.3 – דוגמאות של שימוש בקשרים

נתבונן בשני פסוקים:
א. אני נוסע לים וגם לטיול.
ב. השמש זורחת והשקדיה פורחת.

את שניהם הצגנו בתעתיק לוגי באמצעות הפסוק: a AND b. מחד – איבדנו ממשמעותם המילולית. מאידך – התרכזנו בקשר AND, המאפיין את שניהם. השפה הפורמלית, אשר סימניה הם אותיות לועזיות, קשרים וסוגריים, מכונה בשם **לוגיקה פסוקית** (propositional logic). האותיות מסמנות את הפסוקים, הקשרים משמשים לבניית פסוקים נוספים והסוגריים שומרים על סדר הקדימויות של הקשרים (בסדר זה נדון בהמשך). את הפסוק נסמן באות לועזית קטנה, בין אם הוא אטומי ובין אם לאו; אם הפסוק אכן אטומי, נציין זאת במידת הצורך. את הפסוק $\neg a$ נכנה בשם **שלילה** של פסוק a. את $((\neg a)\vee(a\wedge b))\rightarrow(b\leftrightarrow c)$ נבין כך: לפנינו פסוק, המורכב מפסוקים c,b,a באמצעות הקשרים $\neg,\wedge,\vee,\rightarrow,\leftrightarrow$ ובאמצעות הסוגריים. הקשר $\rightarrow$ בפסוק זה מכונה **הקשר הראשי**. נציין, כי בכל פסוק מספר הסוגריים השמאליים שווה למספר הסוגריים הימניים.

תרגיל 1.2

לפניך פסוקים המסומנים באותיות a, b, c, d. קרא אותם בעיון והשב על שתי השאלות שבהמשך:

a: האביר אוהב את הנסיכה.
b: הרשע חוטף את הנסיכה.
c: האביר מציל את הנסיכה.
d: הנסיכה מתחתנת עם האביר.

א. הפסוקים הבאים מורכבים מפסוקים a, b, c, d ומיוצגים בתעתיק לוגי. כתוב אותם בעברית.

1) $a\wedge c$.
2) $c\rightarrow d$.
3) $(\neg b)\rightarrow(\neg c)$.
4) $(c\wedge b)\leftrightarrow a$.
5) $((\neg a)\rightarrow b)\vee((\neg c)\rightarrow(\neg d))$.

ב. את הפסוקים העבריים שלפניך הצג בתעתיק לוגי בעזרת האותיות , c, d a,b והקשרים המתאימים. היעזר בסוגריים במידת הצורך.

1) האביר מציל את הנסיכה, אם ורק אם הרשע חוטף אותה.
2) אם הרשע אינו חוטף את הנסיכה, והאביר אינו מציל אותה, אז הנסיכה אינה מתחתנת עם האביר.
3) אם האביר אוהב את הנסיכה, אז אם הרשע חוטף את הנסיכה, אז האביר מציל אותה.
4) אם האביר אינו אוהב את הנסיכה, או האביר אינו מציל את הנסיכה, אז הנסיכה אינה מתחתנת עם האביר.

1.3 לוחות-אמת של הקשרים NOT ,OR ,AND

את ערך האמת של פסוק מורכב קובעים מתוך ערכי-אמת של הפסוקים שמהם הוא מורכב ובהתאם להגדרות שלהלן:

הגדרה 1.1

ערך-אמת של פסוק $a\wedge b$ (a AND b) הוא T אם ורק אם לשניהם, ל-a ול-b ערך-אמת T.

את ההגדרה נלווה בלוח המכונה "לוח-אמת של הקשר", אשר מתאר את ערך-אמת של $a\wedge b$ לכל צירוף אפשרי של ערכי-אמת של a ושל b.

a	b	$a\wedge b$
T	T	T
T	F	F
F	T	F
F	F	F

איור 1.4 – לוח-אמת של הקשר $\wedge$

תרגיל 1.3

נסמן ב-a את הפסוק: איתן אוהב את דנה.
נסמן ב-b את הפסוק: איתן אוהב את רותי.
חבר של איתן הודיע לנו כי ערך האמת של הפסוק $a \wedge b$ הוא F. האם ניתן להסיק חד-משמעית מי היא אהובתו של איתן, דנה או רותי? נמק את תשובתך.

הגדרה 1.2

ערך האמת של הפסוק $a \vee b$ (a OR b) הוא T, אם ורק אם, לפחות לאחד מהפסוקים a או b יש ערך-אמת T.

a	b	$a \vee b$
T	T	T
T	F	T
F	T	T
F	F	F

איור 1.5 – לוח-אמת של הקשר $\vee$

תרגיל 1.4

נסמן ב-a את הפסוק: בצפון יורד גשם.
נסמן ב-b את הפסוק: בחרמון יורד שלג.
נמסר לנו מן השירות המטאורולוגי כי ערך האמת של הפסוק $a \vee b$ הוא T וערך האמת של הפסוק $a \wedge b$ הוא F.
מה ניתן להסיק על מזג האוויר בצפון ובחרמון? נמק את תשובתך.

הגדרה 1.3

ערך האמת של הפסוק $\neg a$ (NOT a) הוא T, אם ורק אם, ערך האמת של a הוא F.

a	$\neg a$
T	F
F	T

איור 1.6 – לוח-אמת של הקשר $\neg$

1.4 סדר קדימויות של קשרים, בניית לוח-אמת של פסוק מורכב

בדומה לסדר קדימויות של פעולות החשבון (כפל קודם לחיבור), קיים גם סדר קדימויות של קשרים. הסדר הוא:

קדימות ראשונה	שלילה	$\neg$	NOT
קדימות שניה	גם	$\wedge$	AND
קדימות שלישית	או	$\vee$	OR

איור 1.7 – סדר קדימויות של קשרים

הערה: נקדים ונאמר, כי הקדימות הרביעית היא של הקשר $\rightarrow$ והקדימות החמישית היא של הקשר $\leftrightarrow$.

על-פי סדר קדימויות זה מתפרש הפסוק $\neg a \vee b$ באופן חד-משמעי כך: $(\neg a) \vee b$. פירוש הדבר הוא, שכאשר רוצים לחשב את ערך האמת של הפסוק הזה, יש לחשב תחילה את ערך האמת של שלילת a, ולאחר מכן לעבור אל לוח-אמת של הקשר OR.
נדגים איפוא, את חישוב ערך האמת של הפסוק $\neg a \vee b$. לצורך חישוב זה נבנה לוח הקרוי לוח-אמת של פסוק, אשר בו ירוכזו ערכי-אמת של פסוק מורכב בהתאם לצירופים שונים של ערכי-אמת של פסוקים אטומיים המרכיבים אותו. לוח-אמת של הפסוק $\neg a \vee b$ נבנה על-פי לוחות-אמת של הקשרים NOT ו-OR ובהתאם לשיקולים הבאים:

השורה הראשונה של הלוח:
אם ערך האמת של a הינו T, וערך האמת של b הינו T,
אז על פי לוח-אמת של הקשר NOT, ערך האמת של $\neg a$ הינו F,
לכן על פי לוח-אמת של הקשר OR ערך האמת של $\neg a \vee b$ הינו T.

השורה השניה של הלוח:
אם ערך האמת של a הינו T, וערך האמת של b הינו F,
אז על פי לוח-אמת של הקשר NOT, ערך האמת של $\neg a$ הינו F,
לכן על פי לוח-אמת של הקשר OR ערך האמת של $\neg a \vee b$ הינו F.

באופן דומה נבנה גם את השורה השלישית והרביעית של הלוח.

a	b	$\neg a \vee b$
T	T	T
T	F	F
F	T	T
F	F	T

איור 1.8 – לוח-אמת של $\neg a \vee b$

עתה נשנה את סדר הקדימויות של הקשרים ונעיין בפסוק $\neg(a \vee b)$. על מנת לוודא כי הפסוק הזה שונה מן הפסוק הקודם, נבנה את לוח האמת שלו. את חישוב ערך האמת שלו נבצע באופן הבא: תחילה נחשב את ערך האמת של הפסוק $a \vee b$, ולאחר מכן נחשב את שלילתו של $a \vee b$. נפרט את הצעד הראשון של בניית לוח-אמת של הפסוק $\neg(a \vee b)$.

השורה הראשונה של הלוח:

אם ערך האמת של a הינו T, וערך האמת של b הינו T,
אז על פי לוח-אמת של הקשר OR ערך האמת של $a \wedge b$ הינו T,
לכן על פי לוח-אמת של הקשר NOT ערך האמת של $\neg(a \vee b)$ הוא F.

באופן דומה נבנה גם את השורות הבאות של הלוח.

a	b	$\neg(a \vee b)$
T	T	F
T	F	F
F	T	F
F	F	T

איור 1.9 – לוח-אמת של $\neg(a \vee b)$

תרגיל 1.5

המקור הדליף לעיתון את פסוק a ואת פסוק b (כלומר, הודיע כי ערך האמת של כל אחד מהם הוא T).

לעיתון נודע כי:
– ערך האמת של הפסוק $\neg(a \vee b)$ הוא T.

האם המקור הזה מהימן? נמק את תשובתך.

1.5 לוחות-אמת של הקשרים IF-THEN, IF-AND-ONLY-IF

נעיין בפסוק: "אני הולך לים רק אם יהיה חם". מפרשים אותו כך: "אם יהיה חם, אז אני הולך לים, ואם לא יהיה חם, אז אני לא הולך לים". כדי לבטא את המשמעות הזאת בשפת לוגיקה פורמלית, משתמשים בקשר אם-ורק-אם (IF-AND-ONLY-IF):

אני הולך לים אם-ורק-אם יהיה חם.

הגדרה 1.4

ערך האמת של הפסוק $a \leftrightarrow b$ (a IF AND ONLY IF b) הוא T, אם ורק אם, ערך האמת של הפסוק a זהה לערך האמת של הפסוק b.

a	b	a↔b
T	T	T
T	F	F
F	T	F
F	F	T

איור 1.10 – לוח-אמת של הקשר ↔

הערה: את הקשר "IF AND ONLY IF" רושמים בראשי תיבות כך: IFF. בעברית משתמשים בסימון "אסם" כקיצור לצירוף "אם ורק אם".

תרגיל 1.6

נסמן ב-a את הפסוק: אני חבר הכנסת.
נסמן ב-b את הפסוק: אני דואג לבוחרים שלי.
אחרי בדיקות בשטח התברר, כי ערך האמת של הפסוק $(a \vee b) \leftrightarrow b$ הוא F.
חשב את ערך האמת של הפסוק $a \leftrightarrow b$. נמק את תשובתך.

ההגדרה הבאה דנה בערך האמת של הפסוק $a \rightarrow b$, או במינוח העברי: אם a אז b. בדיבור יום-יומי מוענקת לצירוף זה משמעות של "גרירה", או "קשר סיבתי": אם a נכון, אז גם b נכון. המשמעות הזאת אינה עומדת בקנה אחד עם ההגדרה הפורמלית של ערך-אמת של הפסוק $a \rightarrow b$. מדוע? – משום שהמשמעות הזאת קובעת כי לכאורה ערך האמת של b תלוי בערך האמת של a.

ניעזר בדוגמה הבאה: דני אמר לחברו "אם יהיה חם, אני הולך לים". נסמן ב-a את הפסוק "יהיה חם", וב-b- את הפסוק: "אני הולך לים". ערך האמת של הפסוק b אינו תלוי בערך האמת של הפסוק a ומכאן משתמע שדני יכול ללכת לים, גם אם יהיה קר.

לפני שנעבור להגדרה 1.5, נעיין בהנחיות אלו:

- הפסוק $a \rightarrow b$ הוא פסוק מורכב, הבנוי על פי כללים פורמליים של לוגיקה פסוקית, בעזרת שני הפסוקים a ו-b ובעזרת קשר →.
- ערכי האמת של הפסוקים a ו-b אינם תלויים זה בזה. לכן כל צירוף של ערכי האמת של הפסוקים האלה הוא אפשרי ומתקבל על הדעת.
- בשל היותו פסוק, זכאי $a \rightarrow b$ לתואר T (אמת), או F (שקר).
- תואר זה מוענק לפסוק $a \rightarrow b$ על ידי הגדרה 5 (שלהלן), עבור כל צירוף של ערכי-אמת של הפסוקים a ו-b.

הגדרה 1.5

ערך האמת של הפסוק $a \rightarrow b$ (IF a THEN b) הוא F, אם ורק אם, ערך-אמת של a הוא T וערך-אמת של b הוא F.

הערה: לפעמים מבטאים את הפסוק $a \rightarrow b$ בדרך זו: b IF a.

a	b	$a \rightarrow b$
T	T	T
T	F	F
F	T	T
F	F	T

איור 1.11 – לוח-אמת של הקשר $\rightarrow$

תרגיל 1.7

האב אומר לבנו: "אם תסדר את החדר שלך, נלך לקולנוע".
למחרת הודיע הבן לאביו: "שיקרת לי!" (כלומר ערך האמת של הפסוק שנאמר על ידי האב הוא F).
האם סידר הבן את חדרו? האם הלכו האב ובנו לקולנוע? נמק את תשובתך.

נפרש את הגדרה 1.5 באמצעות שני פסוקים אטומיים:
הפסוק a – אני ראש העיר.
הפסוק b – אני משפר את איכות החיים.

נציג את הפסוק המורכב $a \rightarrow b$ בתעתיק העברי: אם אני ראש העיר, אז אני משפר את איכות החיים.
נוכל לומר, כי ראש העיר (או המעוניין להיות ראש העיר), הוא **דובר אמת**, אם ערך האמת של הפסוק $a \rightarrow b$ הינו T, והוא **אינו אמין**, אם ערך האמת של הפסוק $a \rightarrow b$ הינו F. נקבע את אמינותו של ראש העיר על-פי הגדרה 1.5. את השורה הראשונה של ההגדרה ניתן להבין כך:

– ראש העיר הוא **דובר אמת**, כי הוא מקיים את הבטחתו: הוא ראש העיר, אשר משפר את איכות החיים.

השורה השניה של ההגדרה אומרת:

– ראש העיר **אינו אמין**. הוא אכן ראש העיר, אבל אין הוא משפר את איכות חיינו.

את השורות שלוש וארבע של ההגדרה נפרש כך:

– המועמד לרשות העיר הינו **דובר אמת**, שכן הוא עדיין לא ראש העיר. מותר לו ליהנות מן הספק, שהרי לו היה ראש העיר, יתכן ובאמת היה משפר את איכות חיינו.

בתרגיל שלפנינו נבחן את אמינותו של אחד מנבחרי הציבור בעזרת לוחות-אמת של לוגיקה פסוקית.

תרגיל 1.8

נסמן ב-a את הפסוק: אני חבר הכנסת.
נסמן ב-b את הפסוק: אני דואג לבוחרים שלי.

חבר אחת המפלגות נשא נאום, שבו אמר:
1) $\neg a$
2) $a \rightarrow b$
לאחר בחירתו לכנסת, הוא נשא נאום נוסף:
3) $a \wedge b$

נודע לנו ממקור מוסמך כי ערך האמת של הפסוק a∧b הוא F.

א. הצג בעברית את נאומיו של חבר הכנסת.

ב. מצא את ערך האמת של הפסוק a→b.

נמק את תשובתך.

חשוב לציין, כי כדי לקבוע ערך-אמת של פסוק מורכב על-פי ערכי האמת של הפסוקים שמהם הוא מורכב, אין אנו צריכים "להבין" את הפסוק המורכב או "לחשוב" על משמעותו. את ערך האמת שלו קובעים על פי הגדרות 1.1 – 1.5 ובאופן פורמלי לחלוטין. נציין גם, כי באשר לפסוקים a ו-b שבהגדרות 1.1 – 1.5, אין זה משנה אם פסוקים אלה הם אטומיים או לא. ולכן לוחות-האמת של ההגדרות נקראים על שם הקשר ולא על שם הפסוק.

נוסיף ונאמר, שכפי שבנינו את לוח האמת של פסוק המורכב משני פסוקים, כך גם ניתן לבנות את לוח האמת של פסוק המורכב משלושה ויותר פסוקים. נדגים את בניית לוח האמת של הפסוק a→(b∧c) על פי ערכי האמת של הפסוקים a, b ו-c.

a	b	c	b∧c	a→(b∧c)
T	T	T	T	T
T	T	F	F	F
T	F	T	F	F
T	F	F	F	F
F	T	T	T	T
F	T	F	F	T
F	F	T	F	T
F	F	F	F	T

איור 1.12 – לוח-אמת של a→(b∧c)

תרגיל 1.9

נתונים הפסוקים האטומיים האלה:
– האביר אוהב את הנסיכה.
– הרשע חוטף את הנסיכה.
– האביר מציל את הנסיכה.

בנה לוח-אמת של הפסוק המורכב:
– אם האביר אוהב את הנסיכה, אז אם הרשע חוטף אותה, אז האביר מציל את הנסיכה.

1.6 פונקציות בוליאניות ופעולות בוליאניות

נתבונן בלוח-האמת של הקשר NOT. ערך האמת של הפסוק NOT a תלוי בערך האמת של הפסוק a ולכן, הקשר NOT הוא פונקציה של הפסוק a. את הפסוק a נכנה בשם **ארגומנט**

(argument). הארגומנט a מקבל אחד משני הערכים: T (אמת), או F (שקר). גם ערך הפונקציה NOT הוא T או F , בהתאם לערך הארגומנט.
הקשר AND הוא פונקציה של שני הארגומנטים a ו-b, אשר כל אחד מהם יכול לקבל את הערכים T או F. הפונקציה AND מקבלת גם היא אחד משני הערכים T או F.
פונקציה, אשר כל ארגומנט בה מקבל את הערכים T או F ואשר ערכה גם הוא T או F, נקראת **פונקציה בוליאנית** (boolean function), על שמו של המתמטיקאי האנגלי George Boole.
קוראים, אשר עיסוקם קשור לאלקטרוניקה בוודאי מכירים את הפונקציות האלו.
כמה פונקציות בוליאניות, בעלות שני ארגומנטים ושונות זו מזו ניתן להגדיר? התשובה היא שש-עשרה. קיימים ארבעה צירופים שונים של ערכי אמת של a ושל b. לכל צירוף יכולה הפונקציה לקבל באופן בלתי תלוי את אחד משני הערכים: T או F. ומכאן, שיש לנו 4^2=16 פונקציות בוליאניות שונות בעלות שני ארגומנטים.
מתוך 16 הפונקציות הבוליאניות בעלות שני ארגומנטים הכרנו 4 בלבד:

שם הפונקציה	**הסימון הלוגי של הקשר**
AND	$\wedge$
OR	$\vee$
IMP (IMPlication)	$\rightarrow$
EQU (EQUivalence)	$\leftrightarrow$

נציין ארבע פונקציות בוליאניות נוספות:

פונקצית TR
(TRue - אמת),
אשר ערכיה הם T עבור כל צירוף של ערכי a ו-b.

a	b	TR
T	T	T
T	F	T
F	T	T
F	F	T

פונקצית EXOR
(EXclusive OR - "או מוציא")
אשר היא שלילתה של הפונקציה EQU.

a	b	EXOR
T	T	F
T	F	T
F	T	T
F	F	F

פונקצית NOR
(neither...NOR)
אשר היא שלילתה של הפונקציה OR.

a	b	NOR
T	T	F
T	F	F
F	T	F
F	F	T

פונקצית A
אשר ערכיה זהים לאלו של a.

a	b	A
T	T	T
T	F	T
F	T	F
F	F	F

לכל פונקציה בוליאנית בעלת שני ארגומנטים ניתן להתאים פסוק, המורכב משני פסוקים אטומיים, אשר ערכי האמת שלו יהיו זהים לאלו של הפונקציה. נתבונן למשל בפסוק $(a\wedge b)\rightarrow b$. אנו מציעים לקורא לבנות את לוח האמת של הפסוק הזה ולבדוק אם ערכי האמת של הפסוק זהים לערכים של פונקצית TR.

לפונקציה בוליאנית יש כינוי נוסף: **פעולה בוליאנית**. נציין, כי לכינוי "בוליאני" ולכינוי "לוגי" בהקשר ללוגיקה פסוקית יש אותו מובן. לכן ניתן לומר "פונקציה לוגית" במקום "פונקציה בוליאנית", "ביטוי לוגי" במקום "ביטוי בוליאני" ו-"פעולה לוגית" במקום "פעולה בוליאנית".

תרגיל 1.10

א. בנה את כל 16 הפונקציות הבוליאניות בעלות שני ארגומנטים.
ב. לכל פונקציה בוליאנית אשר בנית, התאם פסוק מורכב בעל ערכי-אמת זהים לאלה של הפונקציה.

לסיום נוסיף את ההערה הבאה:
הקוראים, אשר עיסוקם קשור לנושאי אלקטרוניקה ומעגלים משולבים יודעים, כי כל אחת מתוך 16 הפונקציות הבוליאניות ניתן להציג באמצעות שני קשרים בלבד: NOT ו-AND (או לחילופין: NOT ו-OR). יותר מזה: כל פונקציה בוליאנית ניתן להציג באמצעות קשר אחד בלבד, המכונה NAND (Not AND). משמע, שניתן לבנות שפה פורמלית באמצעות שני קשרים לוגיים בלבד: השלילה $\neg$ והגימום $\wedge$, ואף ניתן לבנות שפה פורמלית באמצעות קשר לוגי אחד ויחיד, אשר לוח-אמת שלו זהה לזה של NAND. מטעמי נוחות וקירבה אל השפה הטבעית העדפנו להשתמש בחמישה קשרים לוגיים. את הדיון בנושא זה נמשיך בפרק הבא.

a	b	NAND
T	T	F
T	F	T
F	T	T
F	F	T

פרק 2

יחסים סמנטיים בין שני פסוקים

סמנטיקה היא המשמעות של מה שנאמר בשפה. כאשר קבענו ערכי-אמת של פסוקים מורכבים, עסקנו בקביעה סמנטית. המושגים מתחום הסמנטיקה, אשר בהם נדון בפרק זה הם:

- פסוקים שקולים,
- טאוטולוגיה,
- סתירה,
- פסוק אשר נובע מפסוק אחר.

2.1 פסוקים שקולים, תכונות השקילות

נעיין בשני הפסוקים האלה:

האחד – $a \to b$

השני – $\neg a \lor b$

מנקודת ראות תחבירית אלה הם פסוקים שונים, אך לא כך הדבר מבחינה סמנטית. מתוך עיון בלוח-אמת משותף של הפסוקים האלה רואים, כי לפסוקים האלה אותו ערך האמת עבור צירופים זהים של ערכי-אמת של a ו-b.

a	b	$a \to b$	$\neg a \lor b$
T	T	T	T
T	F	F	F
F	T	T	T
F	F	T	T

איור 2.1 – שקילות של פסוקים $a \to b$ ו-$\neg a \lor b$

הגדרה 2.1

שני פסוקים הם **שווי-ערך**, או **שקולים**, אם שניהם מקבלים ערך-אמת זהה, עבור כל צירוף של ערכי-אמת של הפסוקים המרכיבים אותם.

נקבל אם-כן, שהפסוק: "אם אני ראש העיר, אז אני משפר את איכות החיים", שקול לפסוק: "אני לא ראש העיר, או שאני משפר את איכות החיים". משמע, ששני הפסוקים האלה הם בעלי תוכן זהה מזווית ראיה של לוגיקה פסוקית.

את שקילות הפסוקים מסמנים בעזרת הסימן $\Leftrightarrow$. קיים קשר הדוק בין הקשר התרבירי $\leftrightarrow$ לבין יחס השקילות $\Leftrightarrow$. הקשר הזה מתבטא כך:

- נניח כי p ו-q הם שני פסוקים מורכבים. אם ערך האמת של הפסוק $p\leftrightarrow q$ הוא T עבור כל צירוף של ערכי-אמת של הפסוקים האטומיים, שמהם מורכבים הפסוקים p ו-q, אזי $p\Leftrightarrow q$.
- נכון גם ההיפך: אם $p\Leftrightarrow q$, אז ערך האמת של הפסוק $p\leftrightarrow q$ הוא תמיד T.

תרגיל 2.1

א. בדוק האם שקולים הפסוקים האלה:
$a\leftrightarrow b$ ו-$(a\rightarrow b)\wedge(b\rightarrow a)$?
$a\leftrightarrow b$ ו-$a\rightarrow b$?
נמק את תשובתך.

ב. הוכח, כי הפסוק "אני אלך לים רק אם יהיה חם" שקול לפסוק "אם יהיה חם, אני אלך לים, ואם לא יהיה חם, אני לא אלך לים".
הוכח, כי הפסוק "אני אלך לים רק אם יהיה חם" **אינו** שקול לפסוק "אם יהיה חם, אני אלך לים".

להלן תכונות השקילות של פסוקים מורכבים:

תכונות החילוף:

א. $a\wedge b \Leftrightarrow b\wedge a$
ב. $a\vee b \Leftrightarrow b\vee a$

תכונות הקיבוץ:

ג. $(a\wedge b)\wedge c \Leftrightarrow a\wedge(b\wedge c)$
ד. $(a\vee b)\vee c \Leftrightarrow a\vee(b\vee c)$

תכונות הפילוג:

ה. $a\wedge(b\vee c) \Leftrightarrow (a\wedge b)\vee(a\wedge c)$
ו. $a\vee(b\wedge c) \Leftrightarrow (a\vee b)\wedge(a\vee c)$

תכונות דה-מורגן (De-Morgan)

ז. $\neg(a\wedge b) \Leftrightarrow (\neg a)\vee(\neg b)$
ח. $\neg(a\vee b) \Leftrightarrow (\neg a)\wedge(\neg b)$

תכונת שלילת השלילה:

ט. $\neg(\neg a) \Leftrightarrow a$

את תכונות השקילות א' – ט' ניתן לבדוק באמצעות שיטות שונות. אחת מהן היא בניית לוחות-אמת משותפים.
נתבונן בתכונות החילוף א' ו-ב'. מבחינה תחבירית הפסוקים $a\wedge b$ ו-$b\wedge a$ שונים זה מזה, אבל מבחינה סמנטית אין הבדל ביניהם: לשניהם יש אותה משמעות, שניהם מוסרים אותו מידע. בהמשך חטיבה זו נתייחס לשניהם כאל אותו פסוק. ההבדל התחבירי שביניהם יהיה רלוונטי, כאשר בחטיבות הבאות נדון ברעיון של תכונות לוגי ויישומו על-ידי שפת התכנות פרולוג.

נעיין בתכונות הקיבוץ ג' ו-ד'. הן מאפשרות להשתחרר מן הסוגריים בפסוקים $a\wedge b\wedge c$ ו-$a\vee b\vee c$, שהרי אין זה משנה מבחינה סמנטית אם בפסוק $a\wedge b\wedge c$ נחשב תחילה את ערך האמת של $a\wedge b$, או את ערך האמת של $b\wedge c$. להבדל התחבירי שבין $(a\wedge b)\wedge c$ לבין $a\wedge(b\wedge c)$ נחזור, כאשר נעסוק בפרולוג.

תרגיל 2.2

מצא פסוקים, השקולים לאלה שלהלן:

א. לא(דובים או יער).
ב. לא(חם או קר).
ג. לא(לא נעים או לא טעים).
ד. לא יתכן שהאביר אינו אוהב את הנסיכה.
ה. לא יתכן שהרשע חוטף את הנסיכה והאביר אינו מציל אותה.

(הערה: בנוגע לשני הפסוקים האחרונים, ראה תרגיל 1.2).

2.2 עושר השפה וקשרים פסוקיים

ישנם לוחות-אמת רבים ושונים. כל אחד מהם מבטא פסוק, אשר יש לו תוכן ומשמעות. את תוכנם ומשמעותם של הפסוקים מבטאים בשפות טבעיות באמצעות חמשת הקשרים אשר היכרנו, ובאמצעות קשרים נוספים כדוגמת:

neither...nor, neither...or, but, all...but ואחרים.

מבחינה תחבירית, עולה עושר הביטוי של שפה טבעית על עושר הביטוי של השפה הפורמלית – שפת לוגיקה פסוקית. לא כך הדבר מבחינה סמנטית. כל פסוק מורכב ניתן לבנות בעזרת שני קשרים בלבד. יותר מזה, ניתן לבנות שפה פורמלית, בה קשר אחד בלבד ישמש את כל לוח האמת. הבעיה היא שביטוייה של השפה הפורמלית הזו ארוכים מאוד ועם סוגריים רבים ולכן, אינם נוחים לקריאה ולהבנה. לכן בחרנו בשפה פורמלית, אשר יש בה קשר פסוקי כנגד הקשרים הנפוצים ביותר בשפות הדיבור הטבעיות.

לקוראים, אשר התנסו בשימוש בפונקציות בוליאניות, נזכיר את פונקצית NAND. באמצעותה ניתן לבטא כל פונקציה בוליאנית אחרת (כפי שהערנו בפרק הקודם). באמצעות הקשר, אשר לוח האמת שלו זהה לזה של NAND, ניתן לבטא כל לוח-אמת. כלומר, ניתן לבנות באמצעותו שפה פורמלית, השקולה לזו שנדונה כאן.

את הנאמר כאן ננסח באופן פורמלי:

- לכל לוח-אמת נתון, מתאים פסוק מורכב, אשר לוח האמת שלו זהה ללוח האמת הנתון.

- כל פסוק מורכב שקול לפסוק, המורכב מאותם פסוקים אטומיים באמצעות שני קשרים בלבד: האחד – קשר השלילה $\neg$, האחר – אחד מהקשרים $\wedge$, $\vee$, $\rightarrow$.

לדוגמה, נעיין בלוח האמת הבא:
(FA משמעו FAlse או FAil)

a	b	FA
T	T	F
T	F	F
F	T	F
F	F	F

לוח האמת הזה מתאים לפסוק המורכב $a\wedge b\wedge(\neg a)$. הוא מתאים גם לפסוק: $(\neg a\vee a)\rightarrow(\neg b\wedge b)$. באשר לפסוק האחרון, נזכירכם כי קשר השלילה קודם לקשרים $\wedge$, $\vee$ (ראה פרק 1) ולכן לא נזקקנו לסוגריים ליד השלילות של הפסוקים.
הפסוק $(\neg a\vee a)\rightarrow(\neg b\wedge b)$ מיוצג באמצעות ארבעת הקשרים: $\neg$, $\vee$, $\wedge$, $\rightarrow$. ניתן להציג אותו באמצעות שני קשרים בלבד: $\neg$, $\vee$. נסמן ב-P את הפסוק הראשון שבקשר $\rightarrow$, וב-Q- את הפסוק השני:

$$P = \neg a\vee a$$
$$Q = \neg b\wedge b$$

צעד ראשון: נתון

$$P\rightarrow Q\Leftrightarrow\neg P\vee Q$$

כלומר:

$$(\neg a\vee a)\rightarrow(\neg b\wedge b) \Leftrightarrow \neg(\neg a\vee a)\vee(\neg b\wedge b)$$

צעד שני: על פי תכונת השקילות ט',

$$\neg(\neg Q) \Leftrightarrow Q$$

כלומר:

$$\neg(\neg(\neg b\wedge b)) \Leftrightarrow \neg b\wedge b$$

צעד שלישי: את הפסוק

$$\neg Q=\neg(\neg b\wedge b)$$

ניתן לשנות כך:

בהתאם לתכונת השקילות ז':

$$\neg(\neg b\wedge b) \Leftrightarrow \neg(\neg b)\vee(\neg b)$$

ולכן בהסתמך על תכונה ט':

$$\neg(\neg b\wedge b) \Leftrightarrow b\vee(\neg b)$$

קיבלנו אם-כן:

$$\neg Q \Leftrightarrow b\vee(\neg b)$$

ולכן:

$$Q \Leftrightarrow \neg(b\vee(\neg b))$$

ולכן:

$$\neg P\vee Q \Leftrightarrow \neg(\neg a\vee a)\vee(\neg(b\vee(\neg b)))$$

בדרך זו הדגמנו, כי את לוח האמת FA ניתן להציג באמצעות הפסוק הבא:

$$(\neg(\neg a\vee a))\vee(\neg(b\vee(\neg b)))$$

אשר מכיל שני קשרים בלבד: $\neg$ ו-$\vee$.

2.3 טאוטולוגיה וסתירה

מבין הקבוצות של פסוקים שקולים ישנן שתי קבוצות מיוחדות. הקבוצה הראשונה מורכבת מפסוקים, אשר לכל אחד מהם יש לוח-אמת זהה לזה של הפונקציה הבוליאנית TR, כדוגמת הפסוק $\neg a\vee a$. פסוק כזה מכונה בשם **טאוטולוגיה**. הקבוצה השניה מורכבת משלילות של טאוטולוגיות. כל אחד מפסוקי הקבוצה הזו מכונה **סתירה**.

הגדרה 2.2

פסוק מורכב, אשר ערך האמת שלו הוא T עבור כל צירוף ערכי-אמת של הפסוקים מהם הוא מורכב, נקרא **טאוטולוגיה** (tautology), או **תקף** (valid).

דוגמאות של טאוטולוגיות:

א) $a\rightarrow a$
ב) $\neg a\vee a$

נניח כי p ו-q הם שני פסוקים שקולים. יוצא אם-כן, כי הפסוק $p\leftrightarrow q$ הוא טאוטולוגיה. את מה שנאמר בסעיף 2.1 על הקשר בין $\leftrightarrow$ לבין $\Leftrightarrow$ ניתן לנסח כך: $p \Leftrightarrow q$ אם ורק אם הפסוק $p \leftrightarrow q$ הינו טאוטולוגיה.

תרגיל 2.3

בדוק אם הפסוק "אני הולך לים וגם לטיול, רק אם אני הולך לטיול וגם לים" הוא טאוטולוגיה, אם לאו.

נסמן ב-true את הפסוק שהוא טאוטולוגיה. בסימון זה אנחנו מתעלמים מן המבנה התחבירי של הטאוטולוגיה ומתרכזים במשמעותה, כלומר בסמנטיקה בלבד. עבור כל פסוק p (מורכב או אטומי) מתקיימות תכונות השקילות הבאות (ראה הגדרה 2.2).

תכונות השקילות (המשך):

י. $p\vee true \Leftrightarrow true$
יא. $p\wedge true \Leftrightarrow p$
יב. $p\rightarrow true \Leftrightarrow true$
יג. $true\rightarrow p \Leftrightarrow p$

הגדרה 2.3

פסוק מורכב אשר ערך האמת שלו הוא F עבור כל צירוף ערכי-אמת של פסוקים שמהם הוא מורכב, נקרא **סתירה** (contradiction).

דוגמאות של סתירות:

א) $\neg a \wedge a$

ב) $\neg a \leftrightarrow a$

את הסתירה נסמן ב-false ונציין את תכונות השקילות שלו.

תכונות השקילות (המשך):

יד.	$p \vee \text{false} \Leftrightarrow p$
טו.	$p \wedge \text{false} \Leftrightarrow \text{false}$
טז.	$\neg(\text{false}) \Leftrightarrow \text{true}$
יז.	$\neg(\text{true}) \Leftrightarrow \text{false}$

תרגיל 2.4

לגבי כל אחד מהפסוקים שלהלן קבע האם הוא טאוטולוגיה, סתירה, או לא זה ולא זה:

א. $\neg a \rightarrow a$

ב. $(a \rightarrow b) \leftrightarrow (b \rightarrow a)$

ג. $(\neg a \rightarrow b) \rightarrow (\neg b \rightarrow a)$

ד. $((a \rightarrow b) \rightarrow (\neg a)) \leftrightarrow (\neg(a \wedge b))$

תרגיל 2.5

קבע לגבי כל אחד מהפסוקים הבאים אם הוא טאוטולוגיה, או סתירה (מנקודת ראות של לוגיקה פסוקית).

א. המכונית נוסעת, או עומדת (כלומר, לא נוסעת).

ב. המכונית נוסעת ועומדת.

ג. המכונית לא נוסעת ועומדת.

ד. המכונית נוסעת ולא עומדת.

רמז: סמן את הפסוק "המכונית נוסעת" ב-a, והצג את פסוקי התרגיל כפסוקים מורכבים.

תרגיל 2.6

התבונן בשני הפסוקים האלה:

א. אם המכונית נוסעת, אז היא עומדת.

ב. אם המכונית נוסעת, אז היא לא עומדת.

תוכל להניח, כי הפסוק "המכונית עומדת" הינו שלילתו של הפסוק "המכונית נוסעת". קבע לגבי כל אחד מהפסוקים שלמעלה אם הוא טאוטולוגיה, סתירה, או לא זה ולא זה.

2.4 יחס של נביעה: פסוק נובע מפסוק אחר

נסמן ב-a את הפסוק: "יהיה חם", ב-b את הפסוק "נלך לים", ונעיין בשני הפסוקים האלה:
האחד – $a \leftrightarrow b$, שמשמעו "נלך לים רק אם יהיה חם".
השני – $a \rightarrow b$, שמשמעו "אם יהיה חם, אז נלך לים".

האם הפסוקים האלה שקולים? נבנה עבור שניהם לוח-אמת משותף:

a	b	$a \leftrightarrow b$	$a \rightarrow b$
T	T	T	T
T	F	F	F
F	T	F	T
F	F	T	T

איור 2.2 – לוח-אמת משותף של הפסוקים $a \leftrightarrow b$ ו-$a \rightarrow b$

ניתן לראות, שכאשר לפסוק הראשון $a \leftrightarrow b$ יש ערך-אמת T, גם לפסוק השני $a \rightarrow b$ יש ערך-אמת T. מאידך, כאשר ל-a ערך-אמת F ול-b ערך-אמת T, ערכי האמת של $a \leftrightarrow b$ ושל $a \rightarrow b$ אינם זהים. לכן הפסוקים האלה אינם שקולים זה לזה.

נעיין בזוג נוסף של פסוקים:
האחד – $\neg a \land a$
השני – $\neg a \land b$

עתה נבנה עבורם לוח-אמת משותף:

a	b	$\neg a \land a$	$\neg a \land b$
T	T	F	F
T	F	F	F
F	T	F	T
F	F	F	F

איור 2.3 – לוח-אמת משותף של הפסוקים $\neg a \land a$ ו-$\neg a \land b$

את שני לוחות האמת האחרונים מאפיינת התכונה המשותפת הבאה: לא קיים צירוף של ערכי-אמת של הפסוקים a ו-b, אשר עבורו לפסוק $\neg a \land a$ יהיה ערך-אמת T ולפסוק $\neg a \land b$ יהיה ערך-אמת F.

הגדרה 2.4

יהיו p ו-q שני פסוקים.
נאמר, כי

הפסוק p **גורר** את הפסוק q,

או:

הפסוק q **נובע** מפסוק p,

אם לא קיים צירוף ערכי-אמת של הפסוקים האטומיים, שמהם מורכבים הפסוקים p ו-q כך, שעבור הצירוף הזה ערך האמת של p הוא T וערך האמת של q הוא F.

את העובדה, כי p גורר את q מסמנים כך:

$$p \Rightarrow q$$

או כך:

$$q \Leftarrow p$$

הערה:
שים לב, כי הגדרה 2.4 אינה דורשת שלפסוק p יהיה אי-פעם ערך-אמת T. את הנאמר בה ניתן לנסח בצורה תופשית, כך: הפסוק p גורר את הפסוק q אם כל עוד ל-p יש ערך-אמת T, גם ל-q יש ערך-אמת T.

דוגמאות:

1. מתוך לוח-אמת המשותף של הפסוקים $a \leftrightarrow b$ ו-$a \rightarrow b$ ראינו, כי הפסוק $a \leftrightarrow b$ גורר את הפסוק $a \rightarrow b$:

$$a \rightarrow b \Rightarrow a \leftrightarrow b$$

יוצא אם-כן, כי הפסוק: "אני הולך לים רק אם יהיה חם" גורר את הפסוק "אם יהיה חם, אני הולך לים".

2. מתוך לוח-אמת של הקשר $\wedge$ רואים, כי לכל זוג הפסוקים p ו-q מתקיים:

$$p \Leftarrow p \wedge q$$

וכמו-כן:

$$q \Leftarrow p \wedge q$$

בהתאם ללוח-אמת של הקשר $\wedge$ לא יתכן, שלפסוק $p \wedge q$ יהיה ערך-אמת T ולאחד מן הפסוקים p או q יהיה ערך-אמת F.
יוצא אם כן, כי:

חם ⇐ חם ולח

וכמו כן:

לח ⇐ חם ולח

3. אם q הוא פסוק השקול לטאוטולוגיה, אז לכל פסוק p מתקיים:

$$p \Rightarrow q$$

במילים אחרות, הטאוטולוגיה נובעת מכל פסוק, שהרי לא יתכן כי לטאוטולוגיה יהיה ערך-אמת F.

כאן המקום לדון שוב בפסוק $p \rightarrow q$. כשבנינו לוח-אמת של $p \rightarrow q$ ציינו את הקושי שבקליטתו והבנתו, בשל המשמעות היום-יומית המוענקת לצירוף "אם-אז". הבה ננתח את היחס שבין

הקשר הפסוקי $\rightarrow$ (חץ ימני) לבין הקשר הסמנטי של גרירה $\Rightarrow$ (חץ ימני מודגש):

– $p\rightarrow q$ הוא פסוק, ובשל כך זכאי לתואר T (אמת) או F (שקר). תואר זה מוענק לו על ידי לוח האמת שלו ומתוך ערכי-אמת של p ו-q. ערכי האמת של p ו-q אינם תלויים זה בזה.

– $p\Rightarrow q$ הוא קשר סמנטי בין פסוק p לבין פסוק q, הקובע כי כל עוד ל-p יש ערך-אמת T, גם ל-q יהיה ערך-אמת T. קשר זה אינו מוסר מידע על ערך-אמת של q, כאשר ל-p ערך-אמת F. ניתן לחשוב על הקשר הזה כעל "תלות חלקית" של q ב-p. q תלוי ב-p כל עוד ערך-אמת של p הוא T ו-q אינו תלוי ב-p, כאשר ערך-אמת של p הוא F.

בדומה לקשר בין $\leftrightarrow$ לבין $\Leftrightarrow$, שנידון בסעיף 2.1, קיים גם קשר בין $\rightarrow$ לבין $\Rightarrow$:
יהיו p ו-q שני פסוקים. נקבל כי $p\Rightarrow q$ אם ורק אם, ערך האמת של הפסוק $p\rightarrow q$ הוא תמיד T.
במילים אחרות: p גורר את q, אם ורק אם, הפסוק $p\rightarrow q$ הוא טאוטולוגיה.

בדיבור היום-יומי אין מבדילים בין הקשר התחבירי $\rightarrow$ לבין היחס הסמנטי $\Rightarrow$ ואת שניהם מבטאים באמצעות הצירוף "אם...אז". כפי שאמרנו, השפה הטבעית אינה חד-משמעית ומדוייקת, להבדיל משפת לוגיקה פורמלית.

נביא דוגמה של דו-משמעותיות, הניתנת לצירוף "אם...אז" בשפה הטבעית.
נעיין בשני פסוקים אלה:
הפסוק a – אני חבר כנסת.
הפסוק b – אני דואג לבוחרים שלי.

איש ציבור אמר בנאומו: אם אהיה חבר כנסת, אדאג לבוחרים שלי. הבוחרים הבינו, כי הובטח להם היחס הסמנטי $a\Rightarrow b$, אשר פוסל את האפשרות שלפסוק a יש ערך-אמת T (משמע, האיש נבחר לכנסת), ולפסוק b יש ערך-אמת F (משמע, הוא איננו דואג לבוחרים). לאחר בחירתו של האיש התברר לבוחריו, כי בנאומו הוא התכוון לקשר התחבירי $a\rightarrow b$, אשר בו ערכי האמת של a ו-b אינם תלוים אחד במשנהו. אם-כן, מבחינתו של איש הציבור, תיתכן האפשרות שערך האמת של הפסוק המורכב $a\rightarrow b$ הינו F כלומר, ערך האמת של a הינו T, וערך האמת של b הינו F.

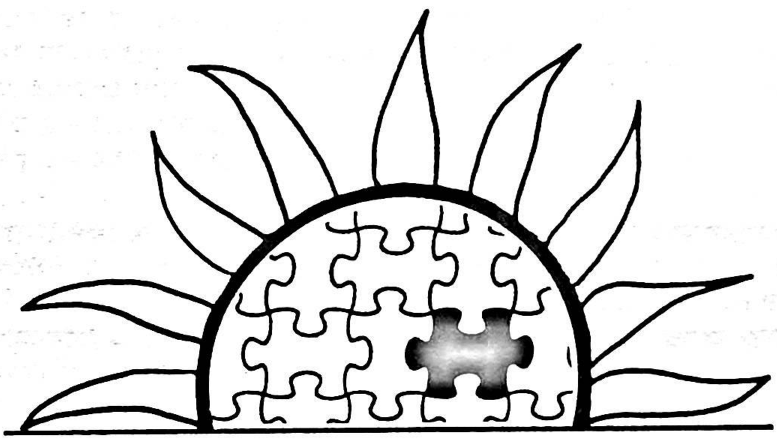
תרגיל 2.7

נתון פסוק אטומי כלשהו a. ענה על השאלות הבאות:
א. האם $a\Rightarrow a$?
ב. האם $a\Rightarrow(a\rightarrow a)$?
ג. האם $a\Rightarrow(a\rightarrow\neg a)$?
נמק את תשובותיך.

תרגיל 2.8

עיין בפסוקים האלה:
א. $a\wedge b$ – אני חבר כנסת, אשר דואג לבוחרים שלי.
ב. $(\wedge a)\wedge(\neg b)$ – איני דואג לבוחרי ואיני חבר כנסת.
ג. $(\neg a)\vee b$ – אני דואג לבוחרים שלי, או שאיני חבר כנסת.
ד. $a\rightarrow b$ – אם אני חבר כנסת, אז אני דואג לבוחרים שלי.
ה. $a\wedge(\neg b)$ – אני חבר כנסת, אשר אינו דואג לבוחרים שלו.

נתח את היחסים הסמנטיים בין הפסוקים א' – ה'.
האם ישנם בתוכם פסוקים שקולים?
האם פסוק כלשהו נובע מפסוק אחר?
נמק את תשובותיך.



פרק 3

יחסים סמנטיים בין פסוק לבין קבוצה של פסוקים

בפרק זה נעסוק ביחסים הסמנטיים שבין פסוק מחד, וקבוצת פסוקים מאידך. לשם כך נחזור ונזכיר את מושגי היסוד של לוגיקה פסוקית, אשר בתחומה אנחנו נמצאים. לאחר מכן נכיר את הבסיס הלוגי של שני מושגים, אשר משתמשים בהם בתחומים ובהקשרים שונים:
– עקביות של קבוצת פסוקים,
– פסוק אשר נובע מקבוצת פסוקים.

3.1 אוסף פסוקים אטומיים של השפה

השפה שאנו עוסקים בה נקראת לוגיקה פסוקית. ביסודה של שפה זו נמצאת קבוצת סימנים המכונים פסוקים אטומיים. באופן אינטואיטיבי אנחנו רואים כל אחד מהפסוקים האטומיים כמייצג פסוק כלשהו בשפה הטבעית (בשפה העברית, למשל). מפסוקים אטומיים בונים פסוקים מורכבים ובדרך זו אנחנו מקבלים את פסוקי השפה, או במילים אחרות, בונים את השפה. את המושג "פסוק" ניתן לתאר כך:
– כל פסוק אטומי הוא פסוק.
– אם a הוא פסוק, אז שלילתו $\neg(a)$ הינה גם כן פסוק.
– אם a ו-b הינם פסוקים, אזי הביטויים שלהלן הינם גם כן פסוקים:

$$(a)\wedge(b)$$
$$(a)\vee(b)$$
$$(a)\rightarrow(b)$$
$$(a)\leftrightarrow(b)$$

הסוגריים נועדו להדגיש, כי הפסוקים a ו-b אינם בהכרח פסוקים אטומיים.

לגבי כל פסוק אנחנו מקבלים את ההנחה הבסיסית הבאה: לכל פסוק יש ערך-אמת T או F, ורק אחד מהם. ערך-אמת של פסוק, המורכב בעזרת הקשרים והסוגריים, נקבע מתוך ערכי-אמת של הפסוקים, אשר מהם הוא מורכב. הקביעה נעשית על פי לוחות-אמת של הקשרים.

הפסוקים a ו-b אשר מופיעים בלוחות-אמת של הקשרים, אינם בהכרח פסוקים אטומיים. נתבונן למשל בפסוק $(a\wedge b)\rightarrow(\neg(a\wedge b))$. בלוח-אמת של הקשר $\rightarrow$ (ראה איור 1.11) נציב את $a\wedge b$ תחת a ואת $\neg(a\wedge b)$ נציב תחת b. בדרך זו נקבל ערכי-אמת של הפסוק $(a\wedge b)\rightarrow(\neg(a\wedge b))$. במקרה הספציפי הזה, הצירופים בהם ערך האמת של $a\wedge b$ זהה לזה של שלילתו $\neg(a\wedge b)$, אינם מציאותיים. כלומר, בלוח האמת אשר התקבל בדרך זו תהיינה שורות מיותרות, אך אין לכך חשיבות בשלב זה.

עקרונית, ניתן לתאר כל פסוק באמצעות פסוקים אטומיים של השפה. נניח למשל כי:

p הוא פסוק $a\wedge b$.
q הוא שלילתו של p.
a ו-b הם פסוקים אטומיים.

אם-כן, הפסוק $p\rightarrow q$ שקול לפסוק $(a\wedge b)\rightarrow(\neg(a\wedge b))$, אשר מורכב מפסוקים אטומיים. נבנה את לוח האמת של הפסוק $p\rightarrow q$ מתוך ערכי האמת של a ו-b, על פי לוחות-אמת של הקשר הראשי $\rightarrow$ והקשרים $\wedge$, $\neg$.

		p	q	$p\rightarrow q$
a	b	$a\wedge b$	$\neg(a\wedge b)$	$(a\wedge b)\rightarrow(\neg(a\wedge b))$
T	T	T	F	F
T	F	F	T	T
F	T	F	T	T
F	F	F	T	T

איור 3.1 – לוח-אמת של הפסוק $p\rightarrow q$ מתוך ערכי-אמת של a ושל b

מלוח האמת ניתן לראות, כי ארבעת הצירופים של ערכי האמת של a ושל b הצטמצמו לשני צירופים שונים בלבד של ערכי-אמת של $a\wedge b$ ושל $\neg(a\wedge b)$.

תרגיל 3.1

יהיו p ו-q הפסוקים שהוזכרו קודם. בנה לוח-אמת של כל אחד מהפסוקים האלה:
א) $p\vee q$.
ב) $\neg p\vee q$.
ג) $q\rightarrow p$.

כל פסוק של השפה מורכב ממספר סופי של פסוקים אטומיים, מספר סופי של קשרים ומספר סופי של סוגריים. לכן, כשנתונה קבוצה סופית של פסוקים, נוכל תמיד "לגלות" את אוסף הפסוקים האטומיים, אשר ממנו נבנו פסוקי הקבוצה.
ניעזר בדוגמה הבאה:

יהיו a, b, p ו-q הפסוקים דלעיל, כלומר, a ו-b הם פסוקים אטומיים, p הוא הפסוק $a\wedge b$ ו-q הוא שלילתו של p. הקבוצה A מורכבת מן הפסוקים הבאים:

$$A=\{p,\ p\vee q,\ p\rightarrow q\}$$

ניתן לראות, שהקבוצה נבנתה משני פסוקים אטומיים: a ו-b. לשם כך נציב את $a\wedge b$ תחת p, ואת $\neg(a\wedge b)$ תחת q.

לגבי פסוק אטומי נקבל את ההנחה הבאה: כל פסוק אטומי יכול לקבל ערך-אמת T או F, ללא קשר לערך-אמת של פסוק אטומי אחר. במילים אחרות, ערכי-אמת של פסוקים אטומיים אינם תלויים זה בזה.

3.2 קבוצה עקבית של פסוקים

באחד מעיתוני הערב נמסרה תחזית מזג-האויר הבאה:

- חם ולח.
- אם חם ולח, אז לא חם או לא לח.

החלטנו לבחון את האמינות של הפסוקים האלה. בניסוח מדויק יותר: החלטנו לבחון אם קיים צירוף של ערכי-אמת של פסוקים אטומיים, שמהם מורכבים הפסוקים לעיל כך, שעבור הצירוף הזה לכל אחד מהפסוקים יהיה ערך-אמת T.
נסמן ב-a את הפסוק האטומי "חם", וב-b את הפסוק האטומי "לח". הפסוק "חם ולח" יסומן כך: $a\wedge b$, והפסוק "לא חם או לא לח" יסומן כך: $(\neg a)\vee(\neg b)$. על פי תכונת שקילות ז' של סעיף 2.1, הפסוק $(\neg a)\vee(\neg b)$ שקול לפסוק $\neg(a\wedge b)$. ניעזר בסימונים של הסעיף הקודם ונסמן ב-p את הפסוק $a\wedge b$ וב-q את שלילתו של p. קיבלנו את קבוצת הפסוקים הבאה:

– חם ולח – p
– אם חם ולח, אז לא חם או לא לח – $p\rightarrow q$

כדי לבדוק אם כל הנאמר באמצעות הקבוצה הוא אמת או לא, נבנה לוח-אמת משותף של קבוצת הפסוקים $\{p,p\rightarrow q\}$.

		p	$p\rightarrow q$
a	b	$a\wedge b$	$(a\wedge b)\rightarrow(\neg(a\wedge b))$
T	T	T	F
T	F	F	T
F	T	F	T
F	F	F	T

איור 3.2 – לוח-אמת משותף של הקבוצה $\{p, p\rightarrow q\}$

בלוח האמת המשותף של הקבוצה, אף לא שורה אחת כוללת ערכי-אמת T לכל אחד מהפסוקים $\{p, p\rightarrow q\}$. משמע, שאין צירוף ערכי-אמת של a ו-b שעבורו לכל אחד מפסוקי הקבוצה יש ערך-אמת T. המסקנה היא, שהקבוצה $\{p, p\rightarrow q\}$ אינה אמינה כתחזית מזג האויר.

לעתים קרובות, בחיי יום-יום, אנו משתמשים במושג "עקביות". אנו נוהגים לומר "התנהגות עקבית", "חוסר עקביות" וכן הלאה.

הגדרה 3.1

תהי A קבוצה (לא ריקה) של פסוקים. נאמר, כי A היא **קבוצה עקבית** (consistent), אם קיים צירוף ערכי-אמת של הפסוקים האטומיים שמהם בנויים פסוקי A כך, שעבור הצירוף הזה יהיה לכל אחד מפסוקי A ערך-אמת T.

אם כן, קבוצת הפסוקים $\{p, p \rightarrow q\}$ של דוגמת תחזית מזג האוויר אינה קבוצה עקבית. נציג דוגמה נוספת של קבוצה לא עקבית. אחד משרי הממשלה, בניסיון לשכנע את העם כי אין להעלות את שכר העבודה, השמיע את הנאום הבא:

- אם יעלה שכר העבודה, אז או שתעלה האינפלציה או שיעלה החוב החיצוני.
- אם החוב החיצוני יעלה וגם האינפלציה תעלה, אז יעלה שכר העבודה.
- אם האינפלציה תעלה, אז שכר העבודה לא יעלה והחוב החיצוני כן יעלה.
- אם לא יעלה שכר העבודה, או אם יעלה החוב החיצוני, אז תעלה האינפלציה.

נבדוק את עקביות נאומו של השר. נשתמש בסימונים הבאים:

a – שכר העבודה יעלה.
b – האינפלציה תעלה.
c – החוב החיצוני יעלה.

הפסוקים שהשר השמיע בנאומו הם אלה:

- $a \rightarrow (b \vee c)$
- $(b \wedge c) \rightarrow a$
- $b \rightarrow (\neg a \wedge c)$
- $(\neg a \vee c) \rightarrow b$

כדי לבדוק את עקביות דבריו של השר (כלומר, עקביות קבוצת הפסוקים שהשמיע) נבנה לוח-אמת משותף של הקבוצה.

a	b	c	$a \rightarrow (b \vee c)$	$(b \wedge c) \rightarrow a$	$b \rightarrow (\neg a \wedge c)$	$(\neg a \vee c) \rightarrow b$
T	T	T	T	T	F	T
T	T	F	T	T	F	T
T	F	T	T	T	T	F
T	F	F	F	T	T	T
F	T	T	T	F	T	T
F	T	F	T	T	F	T
F	F	T	T	T	T	F
F	F	F	T	T	T	F

איור 3.3 – בדיקה של עקביות נאומו של השר

נסקור את השורות בלוח האמת המשותף. כל אחת מהן מציגה צירוף כלשהו של ערכי-אמת של הפסוקים האטומיים. אפשר לראות, כי בכל שורה יש ערך-אמת F לאחד מפסוקי הנאום. באף לא שורה אחת נמצא, כי לכל פסוקי הנאום יש ערך-אמת T. מכאן המסקנה, כי נאומו של השר אינו עקבי.

תרגיל 3.2

בחר מאמר בעיתון (למשל, את מאמר המערכת) ובדוק את עקביות הנאמר בו.

3.3 יחס של נביעה: פסוק נובע מקבוצה של פסוקים

אפשר להניח, שכאשר הציבור שומע נאומים בלתי עקביים של נבחריו הוא "מסיק מסקנות". אם הוא לא מסיק מסקנות, "נובע" מכך, שהוא ציבור אדיש. אנחנו משתמשים במילים: "נובע", "להסיק מסקנות" גם בחיי היום-יום וגם בהוכחות מדעיות. בסעיף זה נדון במשמעות נוספת, אשר ניתנת למילה "נובע" בלוגיקה פסוקית. את המושג "להסיק מסקנות" נכיר בפרק הבא.

הגדרה 3.2

תהיה A קבוצת פסוקים (לא ריקה) ויהי p פסוק. אנו אומרים כי:

הקבוצה A גוררת את הפסוק p,

או:

הפסוק p נובע מהקבוצה A,

אם אינו קיים צירוף של ערכי-אמת של פסוקים אטומיים שמהם מורכבים הפסוק p ופסוקיה של A, אשר עבורו:

ערך-אמת של כל אחד מפסוקיה של A הוא T
וערך-אמת של p הוא F.

בלשון חופשית ניתן לומר, כי p נובע מ-A, אם בכל פעם שלכל אחד מפסוקיה של A יש ערך-אמת T, גם ל-p יש ערך-אמת T.
את העובדה שהפסוק p נובע מקבוצת הפסוקים A מסמנים כך:

$$A \Rightarrow p$$

או כך:

$$p \Leftarrow A$$

נעיר, כי הגדרה 3.2 אינה דורשת שיימצא צירוף ערכי-אמת של פסוקים אטומיים אשר A בנויה מהם, עבורו לכל פסוק של A יהיה ערך-אמת T, כלומר הקבוצה A של ההגדרה 3.2 היא לאו דווקא עקבית.

דוגמאות:

א. נתונה קבוצה A כלשהי, הכוללת שני פסוקים אטומיים a ו-b:

$$A = \{a,b\}$$

נתבונן בלוחות-אמת של הקשרים $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$. ניתן לראות, כי:

$$\{a, b\} \Rightarrow a \wedge b$$
$$\{a, b\} \Rightarrow a \vee b$$
$$\{a, b\} \Rightarrow a \rightarrow b$$
$$\{a, b\} \Rightarrow a \leftrightarrow b$$

נציין, כי A אינה גוררת את הפסוק $\neg a$ וגם לא את הפסוק $\neg b$.

ב. נתונה קבוצה A, המורכבת משני פסוקים p ו-q (לאו דווקא אטומיים):

$$A = \{p, q\}$$

הגדרה 3.2 קובעת כי:

$$\{p, q\} \Rightarrow p \wedge q$$
$$\{p, q\} \Rightarrow p \vee q$$
$$\{p, q\} \Rightarrow p \rightarrow q$$
$$\{p, q\} \Rightarrow p \leftrightarrow q$$

ג. יהיו a ו-b פסוקים אטומיים. נסמן ב-p את הפסוק $a \wedge b$, ב-q- את שלילתו של p, וב-A- את הקבוצה הבאה: $A = \{p, p \rightarrow q\}$.
לוח-אמת משותף של הקבוצה A בנינו בסעיף הקודם.
הגדרה 3.2 קובעת, כי כל פסוק נובע מ-A, שהרי A אינה עקבית ולכן לא קיים צירוף של ערכי-אמת של הפסוקים a ו-b, אשר עבורו ערך-אמת של כל אחד מפסוקיה של A הוא T. לכן, כל פסוק שהוא נובע מ-A, ללא קשר לערך-אמת שלו.
משתמע אם-כן, שתחזית מזג האויר, אשר אותה ניתחנו בסעיף הקודם, מבטיחה לנו גם חום וגם שמש וגם קור וגם שלג – הכל בבת אחת.

ד. נתונה הקבוצה:

$$A = \{p, p \rightarrow q\}$$

נראה כי:

$$A \Rightarrow q$$

נניח, כי ערך האמת של p הוא T וכך גם ערך האמת של $p \rightarrow q$. נבנה לוח-אמת של $p \rightarrow q$. אנו רואים, כי יש שורה אחת בלבד בה יש ל-p וגם ל-$p \rightarrow q$ ערך-אמת T. גם ל-q יש בשורה זאת ערך-אמת T.

p	q	$p \rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

ולכן, הקבוצה A גוררת את הפסוק q.

ה. נתונה הקבוצה:

$$A = \{p \vee q, \neg q \vee u\}$$

אזי:

$A \Rightarrow p \vee u$ (בדוק!)

ו. נתונה הקבוצה: $A = \{\neg q, p \rightarrow q\}$

אזי: $A \Rightarrow \neg p$ (בדוק!)

תרגיל 3.3

נתונה קבוצה A הכוללת שני פסוקים a ו-b:

$$A = \{a, b\}$$

להלן רשימת פסוקים. בדוק כל אחד מהם וקבע, אם הוא נובע מ-A, אם לאו. נמק את תשובתך.

א) a
ב) $a \wedge (\neg b)$
ג) $true$
ד) $false$
ה) $(\neg a) \leftrightarrow b$
ו) $a \rightarrow false$
ז) $\neg a \wedge b$
ח) $(b \rightarrow false) \rightarrow a$
ט) $(\neg b \rightarrow true) \rightarrow a$

3.4 תכונות של יחס הנביעה

נחזור להגדרה 3.2. להלן כמה מהתכונות של הקשר הסמנטי $\Rightarrow$, הנובעות מההגדרה הזאת.

1. אם הפסוק p שייך ל-A, אז הפסוק p נובע מ-A.
הסבר: אם לכל אחד מפסוקי A יהיה ערך-אמת T, אז גם ל-p יהיה ערך-אמת T, שהרי הוא אחד מפסוקיה של A.

2. אם פסוק p הינו טאוטולוגיה, אז הוא נובע מכל קבוצת פסוקים.
הסבר: ל-p יהיה תמיד ערך-אמת T. לכן לא יתכן שלכל אחד מפסוקיה של A יהיה ערך-אמת T ולפסוק p יהיה ערך-אמת F.

3. אם אחד מפסוקיה של A הוא סתירה, אז כל פסוק נובע מ-A.
הסבר: לפסוק שהוא סתירה, תמיד יש ערך-אמת F והקבוצה A מכילה אותו. אי לכך, לא יתכן שלכל אחד מפסוקיה של A יהיה ערך-אמת T. ניקח פסוק p כלשהו, ונפנה לפסוקים האטומיים, המרכיבים אותו ואת הפסוקים של A. האם קיים צירוף של ערכי-אמת של הפסוקים האטומיים האלה אשר עבורו ערך-אמת של כל אחד מפסוקיה של A הוא T וערך-אמת של p הוא F? לא. צירוף כזה אינו קיים, משום שאחד מפסוקיה של A הוא סתירה. לכן על פי הגדרה 3.2, הפסוק p נובע מקבוצה A.

4. אם הקבוצה A כוללת פסוק ושלילתו, אז כל פסוק נובע מ-A.
הסבר: אם לפסוק יש ערך-אמת T, אז לשלילתו יש ערך-אמת F, ולהיפך. לכן, A אינה עקבית. אי-לכך, לא יתכן שלכל אחד מפסוקיה של A יהיה ערך-אמת T. אם נמשיך את ההסבר באופן דומה לזה של תכונה 3, נגיע למסקנה כי כל פסוק נובע מ-A.

5. אם הקבוצה A אינה עקבית, אז כל פסוק נובע מ-A.
הסבר: הקבוצה A אינה עקבית, ולכן לא יתכן שלכל אחד מפסוקיה של A ערך-אמת T. אם נמשיך את ההסבר בצורה דומה לזאת של תכונה 3, נגיע למסקנה כי כל פסוק נובע מ-A.

תכונה 5 קובעת, כי מקבוצה שאינה עקבית נובעים פסוק ושלילתו.
על-פי תכונה 5, מקבוצה, אשר אינה עקבית נובע כל פסוק. בין היתר, נובעים ממנה פסוק ושלילתו. לכן, השר אשר בסעיף 3.2 ניסה לשכנע את העם כי אין להעלות את שכר העבודה, שכנע את העם בשני דברים סותרים, דהיינו: יש להעלות את שכר העבודה ואין להעלותו.

התכונה הבאה היא היפוכה של תכונה 5.

6. אם כל פסוק נובע מ-A, אז A אינה עקבית.
הסבר: כל פסוק ובכלל זה סתירה, נובעים מ-A. לו קבוצה A היתה עקבית, היינו מסיקים, כי קיים צירוף ערכי-אמת של פסוקים אטומיים, שמהם בנויים פסוקי A, עבורו לכל פסוק של A יש ערך-אמת T. לאותו צירוף, גם לסתירה, אשר נובעת מ-A, יש ערך-אמת T, דבר שלא יתכן. לכן A איננה עקבית.

7. אם A הינה קבוצה עקבית ו-p נובע מ-A, אז הקבוצה B, אשר פסוקיה הם פסוק p ופסוקים של A, הינה עקבית גם-כן.
הסבר: A עקבית, ולכן קיים צירוף של ערכי-אמת עבורו לכל פסוק של A יש ערך-אמת T. הפסוק p נובע מ-A, לכן לאותו צירוף גם ל-p יש ערך-אמת T. ומכאן שקיים צירוף של ערכי-אמת עבורו לכל פסוק של B יש ערך-אמת T. לכן B הינה קבוצה עקבית.

לקבוצה B קוראים **איחוד של שתי קבוצות**: האחת מהן A והשניה – קבוצה שמורכבת מאיבר אחד ויחיד {p}. את איחוד הקבוצות מסמנים בעזרת הסימן $\cup$:

$$B = A \cup \{p\}$$

תרגיל 3.4

נתונים הפסוקים הבאים:
פסוק a: יכולה להסתמן אפשרות של דיון על פשרה טריטוריאלית.
פסוק b: יופעלו הסדרי-ביניים בניהול ענייני היומיום במשך תקופה מסוימת (הקרויה גם תקופת ביניים).
במעריב של יום ו', 2 באוקטובר 1987, הופיעה הכותרת הבאה:

1. המזכיר לא הוציא מכלל אפשרות דיון על פשרה טריטוריאלית "בסוף הדרך", אך הדגיש, כי זו יכולה להסתמן רק אחרי שיופעלו הסדרי-ביניים במשך תקופה מסוימת.

2. מדובר בשיתוף ירדן בניהול ענייני היומיום בתקופת הביניים, שאחריה הכל פתוח למו"מ.

עיין בכותרת של מעריב ובצע את המטלות האלה:
א. תרגם את הכותרת לשפה פורמלית של לוגיקה פסוקית, ועל פי התרגום שלך בדוק אם הכותרת הינה עקבית.
ב. בדוק האם נובע מהכותרת של מעריב הפסוק שלהלן:

אם יופעלו הסדרי-ביניים בניהול ענייני היומיום במשך תקופה מסוימת, אז יכולה להסתמן אפשרות של דיון על פשרה טריטוריאלית.

ג. אחרי שניתחת את הכותרת הנ"ל, תאר באופן חופשי את ההבדלים בין שפה טבעית לבין שפה פורמלית. האם לפי דעתך, קיימת שפה נוספת – השפה הדיפלומטית?

ד. בתאריך 7 באוקטובר 1987 בעיתון "חדשות" הופיע הפסוק הבא:

"צריך קודם לקרוא את פסק-הדין, לקיים התייעצויות ולהחליט באיזה טיעונים לנקוט", הסביר פרקליט בכיר, "צריך להבין שהכל כאן כתוב לא בשפה שלנו אלא בשפת הדת, שהיא מרומזת יותר משאומרת בפירוש".

האם לפי מיטב הבנתך, קיימות שפות, כגון: שפה משפטית, שפת הדת, שפה טכנית, וכדומה? אם אכן קיימות, תאר (באופן בלתי פורמלי) את איפיוניהן.



פרק 4

מנגנון ההוכחה של לוגיקה פסוקית

תכנית לוגית היא קבוצת אקסיומות, המאפשרות להוכיח משפטים בעזרת כללי היסק נאותים.
בפרק זה נדון במושגים הבאים:

- אקסיומה,
- משפט,
- כלל היסק,
- הוכחה,
- מערכת היסק.

4.1 כלל היסק נאות, קבוצת אקסיומות, מערכת היסק

תחילה נדון במונח **כלל היסק**. נעיין בקבוצה סופית וסדורה של פסוקים. המשמעות של "סדורה" היא, ששינוי של סדר פסוקי הקבוצה משנה את הקבוצה. לפסוק האחרון יש תפקיד מיוחד. הוא מכונה **התולדה המיידית** של הפסוקים שקדמו לו. הקבוצה עצמה קרוייה **כלל היסק**.
ניעזר בדוגמה הבאה:

$$\{p,\ q,\ p\wedge q\}$$

נכריז שקבוצה זו הינה כלל היסק, כלומר, שהפסוקים p ו-q מולידים באופן מיידי את הפסוק שצורתו התחבירית $p\wedge q$. הפסוקים p ו-q הם פסוקים כלשהם. לכן, על פי כלל ההיסק הזה, הפסוק $(\neg a)\wedge(\neg b)$ הוא התולדה המיידית של הפסוקים $\neg a$ ו-$\neg b$, הפסוק $(a\leftrightarrow b)\wedge(c\vee d)$ הוא התולדה המיידית של הפסוקים $a\leftrightarrow b$ ו-$c\vee d$, וכן הלאה. כלומר, כל פסוק, אשר הקשר $\wedge$ משמש בתור הקשר הראשי שלו, הוא התולדה המיידית של הפסוקים העומדים משני צדיו של הקשר על פי כלל ההיסק הזה.

המונח "כלל היסק" הוא מונח תחבירי בלבד. לכן מותר לנו להכריז, באופן פורמלי לחלוטין, כי הקבוצה $\{p,\ \neg p\}$ היא כלל היסק. כלומר, אנחנו מכריזים כי כל פסוק מוליד את שלילתו באופן מיידי: על פי הכרזה זו, הפסוק "חם" מוליד באופן מיידי את הפסוק "לא חם", והפסוק "אוהב"

– את הפסוק "אינו אוהב". מבחינה פורמלית ההכרזה הזו "שווה" בהחלט, אבל אין אנו מעוניינים בהכרזות סותרות מסוג זה.

כדי "להכניס מוסר" לעניין, ניעזר במונח **נאות** (sound). המוסר – משמעו סמנטיקה, כלומר "משמעות הנאמר". אנו רוצים להקפיד שערך האמת של התולדה המיידית לא "יתנגש" בערכי-אמת של הפסוקים, אשר הולידו אותה.

הדבר מנחה אותנו אל ההגדרה הבאה: אומרים, כי **כלל היסק הוא נאות**, אם התולדה המיידית שלו **נובעת** מקבוצת הפסוקים, אשר הולידה אותה (את המונח **נובע** הכרנו בפרק הקודם). כלומר, הפסוק האחרון של קבוצת פסוקים סדורה, המכונה כלל היסק, **נובע** מקבוצת פסוקים אשר קדמה לו. בלשון אחרת ניתן לומר, כי לכלל היסק נאות יש בסיס לוגי.

לכאורה, פתרנו באמצעות כלל היסק נאות את בעית "מוסר ההכרזות". אלא שלא בדקנו את עקביות הפסוקים, אשר קדמו לתולדה. נתבונן בכלל ההיסק הבא: $\{p, \neg p, q\}$. הכרזנו אם כן, באמצעות הכלל הזה, כי פסוק כלשהו הוא התולדה המיידית של הפסוקים p ו-$\neg p$. כלל ההיסק הזה הוא נאות, שהרי הקבוצה $\{p, \neg p\}$ איננה עקבית, ולכן כל פסוק נובע ממנה. אבל אין אנו מעוניינים בעזרת כלל היסק זה לתת גיבוי למצב, כאשר דבר והיפוכו נובעים מקבוצת פסוקים על בסיס לוגי מוצק של כלל היסק נאות.

נציג מונח חדש: **מערכת היסק** (deductive system). מערכת היסק היא אוסף כללי היסק נאותים ביחד עם קבוצה עקבית של פסוקים, המכונים **אקסיומות של המערכת**. את כללי ההיסק נפעיל רק על האקסיומות של המערכת. פירוש הדבר, כי אם כלל היסק $\{p, \neg p, q\}$ קובע, שפסוק ושלילתו מולידים כל פסוק אחר, אנו צריכים לפנות אל האקסיומות של המערכת ומתוכן למצוא זוג המורכב מפסוק ושלילתו. במצב זה נוכל להסיק, כי כל פסוק הוא תולדה מיידית של מערכת היסק נתונה. אבל, קבוצת האקסיומות היא קבוצה עקבית ולכן אין היא כוללת פסוק ושלילתו. יוצא אם-כן, כי האמצעי הטכני (דהיינו כלל ההיסק $\{p, \neg p, q\}$) אכן קיים, אלא שאין בסיס להפעלתו. אם-כן, פתרנו את בעית "מוסר ההכרזות" באמצעות מערכת היסק.

4.2 משפט והוכחה, מערכת היסק שלימה

ההתנסות שלנו במונח "הוכחה" מתחילה בגיל הרך ואינה מסתיימת בגיל הזהב. בשיעורי המתמטיקה, בבית המשפט ובמקומות רבים אחרים אנו עוסקים באימות טענות בעזרת כלי הקרוי "הוכחה". **הטענה** היא פסוק, אשר ראוי לתואר T (אמת) או F (שקר) **וההוכחה** היא תהליך, אשר סופו אימות הטענה. בתהליך זה אנו משתמשים במערכת היסק, המורכבת מקבוצה עקבית של אקסיומות (המכונות גם הנחות) ומאוסף של כללי היסק נאותים.

תהליך ההוכחה הוא כדלהלן: תחילה, באמצעות כללי היסק של המערכת, רושמים את התולדות המיידיות של חלק מההנחות. בשלב הבא, רושמים פסוקים חדשים, שהם התולדה המיידית של הקבוצה המורחבת. את הפסוקים החדשים מכנים בשם **תולדות** של הקבוצה המקורית (מבלי תוספת "מיידיות"). כך ממשיכים, עד שמגיעים לפסוק, אשר הוא הטענה שהחלטנו להוכיח. ההוכחה מסתיימת כאשר הפסוק הזה נרשם כתולדה של הקבוצה המקורית.

נציין, כי הקבוצה המורחבת היא קבוצה עקבית. מדוע? כי הקבוצה המקורית של ההנחות היא קבוצה עקבית, ולכן יש צירוף של ערכי-אמת, עבורו לכל פסוק של קבוצת ההנחות ערך-אמת T. כלל היסק הוא נאות, כלומר התולדה המיידית נובעת מקבוצת ההנחות, לכן עבור אותו צירוף נקבל לכל אחת מהתולדות המיידיות ערך-אמת T. יוצא אם-כן, שהקבוצה המורחבת

היא עקבית, שהרי יש צירוף שעבורו לכל אחד מפסוקיה יש ערך-אמת T (ראה גם את הוכחת תכונה 7 בסעיף 3.4).

את התהליך הזה ניתן גם לתאר כך: נתונים קבוצת הנחות ואוסף כללי היסק נאותים. באמצעות כללי ההיסק אנו בונים קבוצות עקביות, אשר קבוצת ההנחות המקורית היא חלק מהן. כשמתקבלת קבוצה הכוללת את הטענה, עוצרים את התהליך ואומרים כי הוכחנו את הטענה.
נציין, כי יש אסטרטגיות הוכחה נוספות, אשר כמה מהן נכיר בהמשך.

פסוקים, אשר יש להם הוכחה במערכת היסק נתונה, מכונים **משפטים של המערכת**, או בקצרה, **משפטים**. לתולדה יש הוכחה ולכן – תולדה היא משפט. כפי שציינו, ההוכחה מתנהלת בעזרת כללי היסק **נאותים**, כלומר, באמצעות כללי היסק, אשר התולדות המיידיות שלהם הן אלו שגם **נובעות** בדרך לוגית. מתבקשת השאלה: האם **כל** הפסוקים הנובעים בדרך לוגית מההנחות, הינם גם משפטים? במילים אחרות, האם לכל פסוק, אשר נובע מהנחות של מערכת היסק יש גם הוכחה במערכת זו, כלומר, האם ניתן לקבל את הפסוק מתוך ההנחות של המערכת על-ידי ההפעלה של כללי ההיסק שלה? התשובה היא שלילית. כלומר, ישנן מערכות היסק בהן לא כל פסוק שנובע מהנחות ניתן להוכחה. מערכת היסק, אשר בה ניתן להוכיח כל פסוק הנובע מההנחות, נקראת **מערכת היסק שלימה**. בלשון אחרת: מערכת היסק היא שלימה כאשר כל פסוק הנובע מההנחות שלה, הוא גם משפט שלה.

דוגמה של מערכת היסק לא שלימה

ניעזר בשפה, אשר סימניה הם שני פסוקים אטומיים a ו-b, חמשת הקשרים ¬, ∧, ∨, →, ↔ וסוגריים. כקבוצת ההנחות נגדיר את הפסוק a ואת כל הטאוטולוגיות של השפה. הקבוצה הזו היא עקבית, שהרי לטאוטולוגיה יש תמיד ערך-אמת T, ולכן כל עוד ערך האמת של a הוא T, נקבל ערך-אמת T לכל אחד מפסוקי הקבוצה. בתור כלל היסק נאות נבחר את הקבוצה {p, q, p∨q}. נתבונן בפסוק $b\to a$. פסוק זה נובע מקבוצת ההנחות מכיון, שאם ערך-אמת של a הוא T, אז גם ערך-אמת של $b\to a$ הוא T. אבל הפסוק $b\to a$ אינו משפט של המערכת, שכן בעזרת כלל ההיסק {p, q, p∨q} ניתן מתוך ההנחות לקבל פסוקים, השקולים ל-a או לטאוטולוגיה. הפסוק $b\to a$ אינו טאוטולוגיה ואינו שקול ל-a, ולכן אינו מהווה משפט של המערכת.

באופן מופשט נאמר כך: למערכת היסק לא שלימה אין מספיק כללי היסק כדי להגיע לכל פסוק, שנובע מההנחות שלה.

4.3 דוגמאות של כללי היסק

בסעיף זה נדון בכמה כללי היסק בעלי חשיבות מיוחדת בתכנות לוגי. נזכיר, כי כלל היסק הוא מונח תחבירי ולא סמנטי. מחד, כלל היסק הינו מניפולציה עם פסוקים כדי לקבל פסוק חדש, המכונה תולדה מיידית ומאידך, כלל היסק נאות הוא מונח בו מעורב גם הנושא הסמנטי, דהיינו: **התולדה המיידית** של הפסוקים **נובעת** מהם.

א. modus ponens: {p, p→q, q}

כלל ההיסק modus ponens קובע, כי הפסוק q הוא תולדה מיידית של שני פסוקים שצורתם p ו-$p\to q$. זהו כלל היסק נאות, שהרי אם ערכי-אמת של p ו-$p\to q$ הינם T, אז גם ל-q יש ערך-אמת T, כפי שלמדנו בדוגמה ד' בסעיף 3.4. החשיבות של modus ponens

מוצאת את ביטוייה במשפט הבא:

מערכת היסק היא מערכת שלימה כאשר:

א. קבוצת ההנחות כוללת את כל אחת מהטאוטולוגיות של השפה,

ב. modus ponens נמנה עם כללי ההיסק שלה.

את הוכחת המשפט אפשר למצוא בספרי הלימוד של לוגיקה מתמטית.

ב. modus tolens: $\{p\to q, \neg q, \neg p\}$

כלל ההיסק modus tolens קובע, כי פסוק שצורתו התחבירית $\neg p$, הוא התולדה המיידית של פסוקים שצורתם $\neg q$ ו-$p\to q$. גם הכלל הזה הוא נאות, שהרי אם ל-$p\to q$ ערך-אמת T ול-q ערך-אמת F אז, על פי לוח-אמת של הקשר $\to$ יהיה ל-p ערך-אמת F. ולכן ל-$\neg p$ יהיה ערך-אמת T, משמע – $\neg p$ נובע מ-$\{p\to q, \neg q\}$.

ג. chaining: $\{p\to q, q\to r, p\to r\}$

כלל ההיסק chaining קובע, כי שני פסוקים שצורתם $q\to r$, $p\to q$ מולידים את הפסוק $p\to r$. אנו מציעים לקורא לבנות לוח-אמת משותף של הקבוצה $\{p\to q, q\to r, p\to r\}$ על מנת להשתכנע, כי כלל היסק זה הוא נאות.

ד. resolution: $\{p\vee q, \neg q\vee r, p\vee r\}$

כלל ההיסק resolution קובע, כי כאשר לשני פסוקים יש צורה $p\vee q$ ו-$\neg q\vee r$, הפסוקים q ו-$\neg q$ "מבטלים" זה את זה ונוצר הפסוק $p\vee r$. ניתן לבדוק באמצעות לוח-אמת משותף של $\{p\vee q, \neg q\vee r, p\vee r\}$ כי זהו כלל היסק נאות.

לכלל היסק זה חשיבות רבה בתחום הוכחות משפטים אוטומטיות.

4.4 הוכחה בדרך השלילה

למדנו, כי תהליך ההוכחה של משפט מתבצע על-ידי "צבירת" תולדות מיידיות של הנחות, הוספתן אל הנחות, "צבירת" תולדות של הקבוצה החדשה והוספתן, וכן הלאה, עד אשר אחת התולדות שהתקבלה היא המשפט. קיימת דרך נוספת להוכחת המשפטים והיא **דרך השלילה**.

כדי להוכיח משפט בדרך השלילה, מוסיפים על קבוצת ההנחות את שלילת המשפט כהנחה נוספת ומוכיחים, כי קבוצת ההנחות המורחבת איננה עקבית. אבל קבוצת ההנחות המקורית היתה עקבית ולכן ניתן לפסוק, כי חוסר עקביות נגרם על-ידי שלילת המשפט, כלומר, כל צירוף של ערכי-אמת, אשר עבורו ערכי האמת של ההנחות המקוריות היו T, מזכה את שלילת המשפט בערך-אמת F. מכאן מסיקים, כי צירוף זה מזכה את המשפט בערך-אמת T, ולכן המשפט **נובע** מקבוצת ההנחות.

כדי להוכיח את חוסר העקביות של קבוצת ההנחות המורחבת פונים אל אוסף כללי ההיסק אשר ברשותנו על מנת להסיק, כי פסוק מסוים p ושלילתו $\neg p$ הם תולדה של הקבוצה המורחבת. כללי ההיסק שלנו נאותים, משמע שאם p ו-$\neg p$ הם תולדות של הקבוצה המורחבת, אז p ו- $\neg p$ נובעים מהקבוצה המורחבת, ולכן היא אינה עקבית. בדרך כלל, מחפשים את הפסוק p בין ההנחות המקוריות. כלומר, רוצים לקבל באמצעות כללי היסק, כי שלילתה של אחת ההנחות היא תולדה של הקבוצה המורחבת. בלשון תמציתית ניתן לתאר את ההוכחה בדרך השלילה כך: כדי לומר, כי פסוק מסוים הוא משפט (כלומר תולדה של ההנחות באמצעות כללי היסק נאותים), אנחנו מוכיחים כי שלילתו של הפסוק איננה משפט.

4.5 השימוש בכללי ההיסק להוכחות אקטואליות

לסיום החטיבה הראשונה נציג אוסף שאלות, לעבודה עצמית. בחיי היומיום פותרים לעיתים קושיות לוגיות כדוגמת השאלות שלהלן מבלי לציין באופן מפורש את האקסיומות ואת כללי ההיסק שעל פיהם פעלנו. בשאלות אלו יש לפתור את הקושיות ולנמק את התשובה על ידי ציון מפורש של כללי ההיסק שעל פיהם התקבל הפתרון.

שאלה 1

בכל אחת מהקבוצות שבהמשך יש שתי אקסיומות ושאלה אחת. ענה על השאלה מתוך האקסיומות ועל פי כללי היסק נאותים. בחר אחת מהתשובות האלה:

כן.

לא.

לא ידוע.

התשובה "לא ידוע" הינה למקרה, כאשר אי-אפשר להסיק מהי התשובה הנכונה.

הסבר את בחירתך.

(כדוגמה לשיטת הפתרון האפשרית, נענה על השאלה של קבוצה א').

קבוצה א':

אקסיומות: השקדיה פורחת.

ט"ו בשבט הגיע אם השקדיה פורחת.

שאלה: ט"ו בשבט הגיע?

תשובה: כן.

לא.

לא ידוע.

הסבר: נסמן ב-a את הפסוק 'השקדיה פורחת',

וב-b את הפסוק 'ט"ו בשבט הגיע'.

האקסיומות: $\{a, a \rightarrow b\}$.

השאלה: b

התשובה: כן, על פי modus ponens.

קבוצה ב':

אקסיומות: השקדיה פורחת אם השמש זורחת.

השקדיה לא פורחת.

שאלה: השמש זורחת?

תשובה: כן.

לא.

לא ידוע.

הסבר את התשובה.

קבוצה ג':

אקסיומות: אם יש צמיחה אז יש ברכה.

יש ברכה.

שאלה: יש צמיחה?
תשובה: כן.
לא.
לא ידוע.
הסבר את התשובה.

קבוצה ד':

אקסיומות: חה-חה-חה אם יש בידור.
חה-חה-חה.
שאלה: יש בידור?
תשובה: כן.
לא.
לא ידוע.
הסבר את התשובה.

קבוצה ה':

אקסיומות: חה-חה-חה אם יש בידור.
אין בידור.
שאלה: חה-חה-חה?
תשובה: כן.
לא.
לא ידוע.
הסבר את התשובה.

קבוצה ו':

אקסיומות: הגוף אינו בריא.
הגוף בריא אם הנפש בריאה.
שאלה: הנפש בריאה?
תשובה: כן.
לא.
לא ידוע.
הסבר את התשובה.

שאלה 2

בכל אחת מהקבוצות שלהלן יש שלוש אקסיומות ושאלה אחת. ענה על השאלה בהסתמך על האקסיומות. ציין את אחת משלוש התשובות האלה:
כן.
לא.
לא ידוע.
הסבר את תשובתך.

קבוצה א':

אקסיומות: יש חיוך אם יש מצב-רוח.
יש מצב-רוח אם יש שמש.
יש שמש.
שאלה: יש חיוך?
תשובה: כן.
לא.
לא ידוע.
הסבר את תשובתך.

קבוצה ב':

אקסיומות: אם יש עבודה, אז יש פרנסה.
אם יש פרנסה, אז יש מצב-רוח.
אין מצב-רוח.
שאלה: יש עבודה?
תשובה: כן.
לא.
לא ידוע.
הסבר את תשובתך.

קבוצה ג':

אקסיומות: אם יש צמיחה, אז יש עבודה.
אם יש עבודה, אז אין בריאות.
יש בריאות.
שאלה: יש צמיחה?
תשובה: כן.
לא.
לא ידוע.
הסבר את תשובתך.

קבוצה ד':

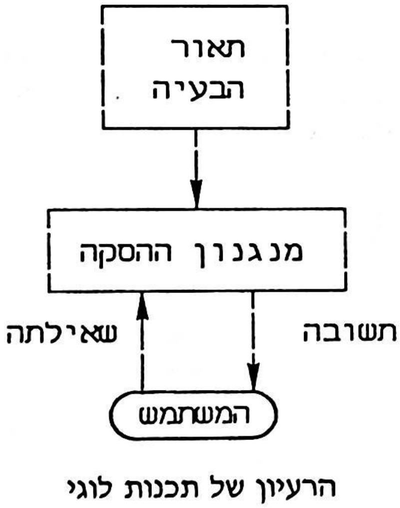
אקסיומות: הגוף בריא אם הנפש בריאה.
הנפש בריאה אם אין מלחמה.
הגוף אינו בריא.
שאלה: יש מלחמה?
תשובה: כן.
לא.
לא ידוע.
הסבר את תשובתך.

חטיבה שניה –

עקרונות של תכנות לוגי

את הרעיון של תכנות לוגי ניתן להציג באופן הבא:

* נתאר את הבעיה.
* נשלב את התאור הזה עם מנגנון ההסקה, אשר אינו תלוי בבעיה.
* נציג שאילתה ונקבל תשובה.



הרעיון של תכנות לוגי

בחטיבה זו נכיר את דרך החשיבה של תכנות לוגי, ונלמד כללים לתאור הבעיה. את הבעיה מתארים בשפה פורמלית, אשר בנויה על בסיס של לוגיקה ומכילה פסוקים משני סוגים: עובדות וחוקים. העובדה היא פסוק, אשר לא ניתן לפרקו לפסוקים פשוטים יותר מבלי לאבד את משמעותו. את העובדה נכיר בפרק 5. מבחינה תחבירית העובדה הינה נוסחה אטומית של לוגיקת היחסים; על כך נדון בפרק 6. החוק הינו פסוק, אשר מורכב מפסוקים פשוטים

באמצעות הקשרים הלוגיים IF, AND, OR, NOT. מבנה של חוק נכיר בפרק 7.

העובדה מתארת יחס בין העצמים, אשר לעיתים אינם ידועים. תהליך הגילוי שלהם הינו חלק של הבעיה. את העצמים שאינם ידועים, מסמנים באמצעות משתנים, כפי שנלמד בפרק 5. נציין, כי משתנים לוגיים הינם מכומתים, משמע, בנוסף לעצם בלתי ידוע, הם מבטאים ערך כמותי, הקשור לעצם זה. בשאלות הכימות נדון בפרק 6.

העובדות והחוקים מבטאים מידע, הרלוונטי לתאור הבעיה. על בסיס מידע זה ניתן להציג שאילתות ולהסיק מסקנות. מנגנון ההסקה מיושם באמצעות תוכנה (בספר זה נשתמש בתוכנה של WISDOM-PROLOG), אשר די לנו להכירה באופן כללי לצורך תכנות לוגי. את תהליך הסקת המסקנות ניתן להסביר במונחים לוגיים, כפי שנביא בפרק 7.

מנגנון ההסקה מופעל על ידי השאילתה. בפרק 5 נדון בשאילתה של תכנות לוגי.

המינוח של תכנות לוגי טרם התמסד בשל גילו הצעיר. אי-לכך, משתמשים במונח "תכנית לוגית" כדי לציין אוסף של עובדות וחוקים, המתארים בעיה. השמות הנוספים לאוסף זה הינם:

* מסד נתונים, או בסיס נתונים – כשהאוסף מכיל עובדות בלבד.
* מסד ידע, או בסיס ידע – כאשר בנוסף לעובדות, האוסף מכיל גם חוקים.

בשם "תכנית לוגית" משתמשים גם כדי לציין את האוסף שמכיל עובדות, חוקים **ושאילתה**, אשר באמצעותה מופעל מנגנון ההוכחה. במקרה זה אומרים: "התכנית רצה", "התכנית מבצעת" וכדומה. להבא, נשתמש במונח "תכנית לוגית" לציון שני מבנים אלה. לאיזה מן המבנים אנו מתכוונים: למסד נתונים, למסד ידע או ל"תכנית רצה", יהיה ברור מההקשר.

את התכניות הלוגיות של החטיבה הזאת ניישם באמצעות שפת התכנות פרולוג בגירסה WISDOM-PROLOG.

פרק 5

עובדות, משתנים ושאילתות

בפרק זה נכיר את המושגים הבאים:

עובדה – מתארת יחסים בין עצמים.
שאילתה – מאחזרת מידע מתוך העובדות שבבסיס הנתונים.
משתנה – מציין עצמים, אשר אינם ידועים.

5.1 יחסים וקבוצות סדורות של עצמים

נתבונן בפסוק: לונדון היא בירת אנגליה. הפסוק הזה מבטא **יחס** בין שני **עצמים**. היחס הוא: בירה של, והעצמים הם: לונדון, אנגליה. סוג יחס זה נקרא דו-מקומי, כיון שהוא יחס המתקיים בין שני עצמים. להלן דוגמאות נוספות של יחסים דו-מקומיים.

עצם ראשון	יחס	עצם שני
אברהם	אב של	יצחק
יצחק	אב של	יעקב
צמיג	חלק של	גלגל
גלגל	חלק של	מכונית
חמש	גדול מ-	שלוש
שלוש	גדול מ-	אחד

זוג העצמים (אברהם, יצחק) עומד ביחס הנקרא אב של. זוג עצמים העומד ביחס דו-מקומי הוא **זוג סדור**. למה סדור? כי אם הופכים את סדר העצמים עלול היחס לחדול מלהתקיים. לדוגמה: אברהם הוא האב של יצחק, אבל יצחק אינו אביו של אברהם. כלומר, הזוג (אברהם,יצחק) והזוג (יצחק, אברהם) אינם זהים בהקשר ליחס אב של.

להלן נציג את היחס ואת העצמים בדרך, אשר שפת הפרולוג אימצה אותה ואשר מקורה בלוגיקת היחסים (predicate logic). את העצמים נכתוב בין סוגריים בהתאם לסדר שנקבע, ואת היחס נכתוב לפני הסוגריים. לא נשאיר רווח בין היחס והסוגר הפותח ונוותר על מילים כמו: של, מ-(כמובן בתנאי שהוויתור לא יפגע במשמעות הנאמר). נציג בצורה זאת את הדוגמאות שהצגנו קודם:

```
אב(אברהם, יצחק)
אב(יצחק, יעקב)
חלק(צמיג, גלגל)
חלק(גלגל, מכונית)
גדול(חמש, שלוש)
גדול(שלוש, אחד)
```

בהמשך נכנה את היחס בשם **פרדיקט** (predicate) ואת העצמים שבסוגריים – בשם **ארגומנטים** (arguments). מקורם של השמות האלה בלוגיקת היחסים. בשמות אלה משתמשים גם בפרולוג.

בנוסף ליחסים הדו-מקומיים ניעזר גם ביחסים תלת-מקומיים, ארבעה-מקומיים, חמישה-מקומיים, וכן הלאה. נעיין בפסוק: צמיג וגלגל הם חלקי המכונית. היחס חלקים של מתקיים בין שלושת העצמים: צמיג, גלגל, מכונית. הקבוצה (צמיג, גלגל, מכונית) היא **שלשה סדורה** בהקשר ליחס זה. היחס חלקים של מתקיים גם בין העצמים של השלשה הסדורה (יד, ראש, גוף). היחס הורים של מתקיים בין העצמים של השלשות הסדורות (אברהם, שרה, יצחק) ו-(דני, ריקי, יוסי). נארגן את היחסים של השלשות באמצעות הכתיב הפורמלי של לוגיקת היחסים, כלומר באמצעות פרדיקטים, ארגומנטים וסוגריים.

```
חלקים_של(צמיג, גלגל, מכונית).
חלקים_של(יד, ראש, גוף).
הורים_של(אברהם, שרה, יצחק).
הורים_של(דני, ריקי, יוסי).
```

אומרים, כי צמיג וגלגל הם חלקי המכונית. אפשר גם לומר: החלקים של המכונית הם הצמיג והגלגל. נתבונן ברשימה הבאה:

```
חלקים_של(מכונית, צמיג, גלגל).
חלקים_של(גוף, יד, ראש).
הורים_של(יצחק, אברהם, שרה).
הורים_של(יוסי, דני, ריקי).
```

ברשימה האחרונה סדר העצמים בין הסוגריים שונה מזה של הרשימה הקודמת. למרות שינוי הסדר ברור, שבכל אחת מן הרשימות הכוונה היא, כי היד והראש הן החלקים של הגוף, ולא אחרת. לגבי אברהם, שרה ויצחק אנו מניחים, כי מדובר במשפחה התנ"כית המפורסמת, ולכן בכל אחת מן הרשימות אברהם ושרה הם ההורים של יצחק. לגבי השלשות (דני, ריקי, יוסי) של הרשימה הראשונה ו-(יוסי, דני, ריקי) של הרשימה השניה, מניחים כי דני וריקי הם ההורים של יוסי, שהרי בכל אחת מן השלשות דני וריקי "תופסים" את מקומותיהם של אברהם ושרה, ויוסי "תופס" את מקומו של יצחק.

כדי להצפין מידע על היחסים והעצמים בצורה ברורה וחד-משמעית ובאמצעות הכתיב הפורמלי של לוגיקת היחסים, חייבים לציין באופן מפורש את תפקידו של כל אחד מארגומנטים. נעיין ברשימה שלהלן:

```
הורים_של(גלית, טל, רן).
הורים_של(לילית, שרית, דן).
הורים_של(חדווה, רוני, משה).
```

הרשימה הזו הינה נטולת מידע ולכן גם נטולת משמעות. כדי להבהיר את הנאמר בה, נוסיף את השורה הבאה:

```
הורים_של(אם, ילד, אב).
```

נסכים, כי בכל אחת משורות הרשימה מבנה הנתונים הוא כזה: במקום הראשון נמצאת האם, במקום השני נמצא הילד ובמקום השלישי האב. נאמר בדרך משל, כי "סדר סדר תרדוף ובסדר תשפוט את רשימתך".

5.2 עובדות, בסיסי נתונים ושאילתות

בפרק זה נכיר כמה מן הרעיונות המהווים את הבסיס של תכנות לוגי ושל שפת הפרולוג. את העולם אשר סובב אותנו מתארים בפרולוג באמצעות הכתיב הפורמלי של לוגיקת היחסים, כלומר, באמצעות היחס וקבוצת העצמים הסדורה. את העובדה כי לונדון היא בירת אנגליה מבטאים בפרולוג כך:

בפרולוג העברי	בפרולוג האנגלי
בירה(לונדון, אנגליה)	capital(london, england)

ה-capital של הפרולוג האנגלי וה-בירה של הפרולוג העברי הם היחסים. העצמים מוקפים בסוגריים ומופרדים בפסיקים. הביטוים לונדון, אנגליה, london, england הינם עצמים.
את העובדה כי אברהם ושרה הם ההורים של יצחק נבטא כך:

הורים_של(אברהם, שרה, יצחק)	parents_of(abraham, sarah, isaac)

ברשימת העצמים נציין במקום הראשון את שם האב, בשני – את שם האם ובשלישי – את שם הילד. בכל מקרה עלינו לשמור על סדר קבוע כפי שהוגדר.

להלן כמה מכללי התחביר של שפת פרולוג:

1. שמות היחסים והעצמים מתחילים **באות**. בפרולוג האנגלי תהיה זו אות אנגלית **קטנה**. את תפקיד האות הגדולה בפרולוג נכיר בהמשך.

2. שמות היחסים והעצמים אינם כוללים רווחים. את הרווחים "סוגרים" באמצעות **קו-תחתון** ("_").

3. אין משאירים רווח בין היחס לבין הסוגר הימני הפותח.

4. כל פסוק של פרולוג (עברי, או לועזי) מסתיים **בנקודה**.

לתכנית הלוגית הראשונה שנכיר נקרא "המשפחה", או The Family. התכנית הזאת היא אוסף של עובדות על המשפחה התנ"כית המפורסמת.

`parent(abraham, isaac).`	הורה(אברהם, יצחק).
`parent(isaac, jacob).`	הורה(יצחק, יעקב).
`parent(sarah, isaac).`	הורה(שרה, יצחק).
`parent(rivka, jacob).`	הורה(רבקה, יעקב).
`male(abraham).`	זכר(אברהם).
`male(isaac).`	זכר(יצחק).
`male(jacob).`	זכר(יעקב).
`female(sarah).`	נקבה(שרה).
`female(rivka).`	נקבה(רבקה).

תכנית 5.1 – The Family (בגירסה עברית ואנגלית)

התכנית The Family מתארת את היחסים האלה: הורה, זכר, נקבה. היחס הורה הוא יחס דו-מקומי בין שני העצמים אשר הראשון בהם הוא ההורה והשני הוא הילד. היחס זכר מכונה יחס **חד-מקומי** או **תכונה**. הכינוי האחר ליחס של פרולוג הוא **פרדיקט**. את העצם מכנים בשם **ארגומנט**. מקורם של הכינויים האלו בלוגיקת היחסים.

התכנית The Family בנויה כולה מעובדות. **עובדה של פרולוג** היא פסוק, אשר לא ניתן לפרקו לפסוקים פשוטים יותר מבלי לשנות את משמעותו (בלוגיקה פסוקית קראנו לפסוק מסוג זה פסוק אטומי). ערך-אמת של עובדה, המופיעה בתכנית, הוא T (אמת) במסגרת אותה תכנית. פירוש הדבר הוא, כי ערך האמת של העובדה הורה(אברהם, יצחק) הוא T (אמת), מכיון שהיא שייכת לתכנית, וערך האמת של העובדה הורה(אברהם, ישמעאל) הוא F (שקר), כי היא אינה שייכת לתכנית. נציין, כי ערך האמת של העובדה הורה(יצחק, אברהם) הוא F (שקר). משמע, שקבוצת העצמים היא קבוצה סדורה וכאשר משנים את סדר העצמים, מפסיק היחס להתקיים.

התכנית The Family היא אוסף של נתונים על המשפחה התנ"כית. ניתן אם-כן לכנותה בשם **בסיס נתונים** (database). זהו מאגר של נתונים, אשר ממנו אפשר לאחזר מידע על המשפחה התנ"כית. בפרולוג ניתן לאחזר את המידע באמצעות **שאילתה**.

המבנה התחבירי של השאילתה זהה לזה של העובדה. ההבחנה בין עובדה לשאילתה נעשית בכל גירסה של פרולוג בדרך האופיינית לה. אנו נציין את השאילתות של פרולוג על ידי סימן שאלה, מקף ורווח אחד לפחות לפני הביטוי (מימין או משמאל, בהתאם לכיוון השפה). לדוגמה:

```
?- parent(abraham, isaac).          ?- הורה(אברהם, יצחק).
yes                                 כן
?- parent(abraham, ismael).         ?- הורה(אברהם ישמעאל).
no                                  לא
?- parent(isaac, abraham).          ?- הורה(יצחק, אברהם).
no                                  לא
?- female(sarah).                   ?- נקבה(שרה).
yes                                 כן
?- female(leah).                    ?- נקבה(לאה).
no                                  לא
```

הצגת שאילתות וקבלת תשובות דומה לשיחה, אשר מתקיימת בין שניים. האחד הוא אדם, והשני אינו אדם. השני הוא מחשב, וזאת יש לזכור היטב. פרטים רבים אין צורך לבטא בשיחת בני-אדם. השם לאה למשל, הוא שם של אישה ובהקשר למשפחה התנ"כית הוא שם של אחת מנשות המשפחה. זהו **ידע שגור** (common sense)- נחלת כל אדם מן הישוב. הוא כולל אין ספור קשרים אסוציאטיביים ונרכש במשך שנים רבות.
הקניית ידע שגור למחשב היא אחת הבעיות הקשות ביותר בתחום של בינה מלאכותית. פרולוג הינה שפת תכנות, כלומר, שפה פורמלית, אף-על-פי שהיא נראית כמעט כמו שפה טבעית. הדמיון בין פרולוג לבין שפה טבעית מטעה לפעמים את המשתמשים בפרולוג. בשל דמיון זה, הם מייחסים למערכת הפרולוג תכונות אנושיות ומצפים לתשובות חיוביות המתבססות על ידע שגור. בהמשך נלמד את כללי ההיסק ואת החוקים, אשר באמצעותם מרחיבה המערכת של פרולוג את מאגר הידע שלה. בכל מקרה, אין להקנות לשפת הפרולוג תכונות של בני-אנוש. יש לנסח את השאילתות על פי כללי התחביר שלה, ויש לצפות לתשובות בהתאם לנתונים שנצברו בבסיס הנתונים.

5.3 משתנים לוגיים, שאילתות עם משתנים

באופן עקרוני, תכנות לוגי הינו תכנות הצהרתי ואינו תכנות פרוצדורלי. תכנית לוגית אידיאלית היא אוסף של עובדות וחוקים, המתארים את המושגים שבהם אנחנו דנים. קל מאוד להבין את הנאמר בה. נעיין בתכנית לוגית 5.2, המכונה Dictionary. זהו מילון אנגלי-עברי. המילה העברית מופיעה בו בתעתיק הלועזי, כיון שהפרולוג האנגלי איננו מכיר את האלף-בית העברי.

```
means(father, av).             begins(av, alef).
means(nurse, achot).           begins(tayas, tet).
means(sister, achot).          begins(beitsa, bet).
means(fly, zvuv).              begins(dod, dalet).
means(stone, even).            begins(even, alef).
means(egg, beitsa).            begins(achot, alef).
means(onion, batsal).          begins(zvuv, zain).
means(uncle, dod).             begins(batsal, bet).
means(pilot, tayas).           begins(tas, tet).
means(fly, tas).
```

תכנית 5.2 – Dictionary

לצורך התרגום השתמש כותב התכנית ביחס means, וכדי למצוא בספר את התעתיק העברי של המילה הוא נעזר ביחס begins.
למעשה, כותב התכנית בנה מאגר מידע, אשר ניתן להשתמש בו לצורך התרגום. כאן הוא זקוק לסוג שאילתות, השונות מאלו, אשר ראינו בפרק הקודם. הוא זקוק לשאילתה שמשמעותה: "מהו התרגום לעברית של המילה father?" הוא גם זקוק לשאילתה: "מהו תרגומה לאנגלית של המילה av?". השאילתה מן הסוג, אשר הופיע בפרק הקודם איננה מתאימה לו.
אם הוא ישאל: ?-means(father, av)
פירוש הדבר שהוא יודע, כי תרגומו של father הינו av.
סוג השאילתה החדש שנלמד הוא **שאילתה עם משתנה**. את המשתנה של פרולוג (עברי ואנגלי כאחד) מסמנים באמצעות אות אנגלית גדולה. אם-כן, את השאלה: "מהו התרגום לעברית של המילה father?" מבטאים בפרולוג כך:

```
?- means(father, X).
```

התשובה של פרולוג תהיה:

```
X = av
```

משתנים של פרולוג הם משתנים לוגיים ועל כך עוד נדון בהמשך. נציין רק כי הם שונים לחלוטין ממשתני התכנות הפרוצדורלי, אשר תפקידם היה לזהות את כתובות הזכרון של המחשב.
השאילתה הבאה שלנו תהיה:

```
?- means(X, av).
```

התחום של המשתנה הוא פסוק אחד בלבד, ולכן המשתנה X של השאילתה השניה איננו קשור למשתנה X של השאילתה הראשונה. התשובה של פרולוג תהיה:

```
X = father
```

כשמעוניינים לקבל תשובות נוספות, מקישים מייד אחרי התשובה את הסימן ";" (נקודה-פסיק).
נעיין בדוגמה שלהלן:

```
?- means(X, achot).
X = nurse;
X = sister;
no (more answers)
```

להלן כמה דוגמאות נוספות של שאילתות ותשובות:

```
?- means(fly, X).
X = zvuv;
X = tas;
no (more answers)

?- means(mother, X).
no
```

התשובה לשאילתה האחרונה היא שלילית. פירושה, כי בסיס הנתונים Dictionary אינו כולל את התרגום לעברית של המילה mother.
ניתן גם להציג שאילתה מן הסוג הבא:

```
?- means(X, Y).
```

בתשובה נקבל את כל זוגות העצמים, המקיימים את היחס means:

```
X = father
Y = av;

X = nurse
Y = achot;

X = sister
Y = achot;
```

וכך הלאה.

הצגה של שאילתה עם משתנה ניתן לפרש כהצגה של שלוש קושיות:

1. האם קיימת הצבה תחת המשתנה עבורה ערך האמת של הפסוק יהיה T (אמת)?

2. מהי ההצבה הזו (בתנאי שהיא קיימת)?

3. האם קיימות הצבות נוספות, ומה הן?

אם התשובה לקושיה הראשונה היא שלילית, נקבל: no (לא). אם התשובה לקושיה הראשונה היא חיובית, נקבל את ההצבה, וכאשר נקיש ";" נקבל הצבות נוספות, אם אלו אכן קיימות. ישנן גירסאות של פרולוג בהן בתשובה לשאילתה מתקבלות ברצף כל ההצבות הקיימות.

סוג אחר של שאילתה עם משתנה הוא שאילתה עם משתנה, הקרוי **משתנה אנונימי** (anonimous variable). הצגת שאילתה מסוג זה מתפרשת כהצגת קושיה ראשונה בלבד. סימונו התחבירי של משתנה אנונימי הוא קו-תחתון, והתשובה המתקבלת היא: yes (כן), או: no (לא).

```
?- means(_, av).
yes
?- means(mother, _).
no
?- means(_, _).
yes
```

5.4 העובדות והשאילתות – תרגול

הגענו עתה לנקודת ביקורת, אשר תוכל לסייע לך לבדוק את שליטתך בחומר שלמדת בסעיפים 5.1-5.3.

הכן עט ונייר. אין צורך במחשב.
עבודה מהנה!

שאלה 1

לפניך פסוקים בשפת הפרולוג. תרגם אותם לעברית או לאנגלית. תן הסברים, אם נראה לך כי התרגום איננו חד-משמעי.

הערה: הפסוקים אינם תלויים זה בזה. כל פסוק הוא תכנית לוגית העומדת בזכות עצמה.

א. `likes(tom, mary).`
ב. `teaches(tim, jack).`
ג. `teacher(tim).`
ד. `teaches(ann, liz).`
ה. `teacher(liz).`
ו. `sisters(mary, ann).`
ז. `father(tim, tom).`
ח. `family(tim, tom, jack).`

שאלה 2

לפניך שתי תכניות לוגיות. תאר מה הן התכונות ומי הם העצמים בכל אחת מהתכניות.

תכנית א'

```
person(old).
person(young).
person(nice).
person(genius).
```

תכנית ב'

```
old(person).
old(house).
nice(person).
nice(weather).
```

שאלה 3

תרגם לפסוקים של הפרולוג האנגלי את המשפטים הבאים:

א. האיש הזה זקן.
ב. האיש הזה חביב.
ג. הבית הזה ישן.
ד. מזג האויר נחמד.

לעזרתך – מילון עזר:

איש – person
זקן – old
ישן – old
חביב – nice
נחמד – nice

שאלה 4

התבונן בתכנית לוגית 5.2, המכונה "Dictionary" (סעיף 5.3) ומצא את כל התשובות האפשריות לשאילתות האלו:

א. `?- means(uncle, _).`
ב. `?- means(brother, _).`
ג. `?- begins(_, alef).`
ד. `?- begins(_, gimel).`
ה. `?- begins(_, _).`
ו. `?- begins(X, bet).`
ז. `?- begins(tas, X).`
ח. `?- begins(X, Y).`

שאלה 5

לפניך בסיס הנתונים:

```
studies(dani, prolog).
studies(ran, prolog).
studies(nili, history).
studies(ran, history).
studies(nili, prolog).
studies(dani, matematics).
studies(ran, matematics).
```

הצג את השאלות א' – ו' בשפת פרולוג, בדומה לדוגמה שלפניך.
דוגמה: מי לומד פרולוג? ?-studies(X, prolog).

א. האם מישהו לומד היסטוריה?
ב. מי לומד מתמטיקה?
ג. האם דני לומד משהו?
ד. מה לומדת נילי?
ה. האם הלימודים מתקיימים? (במילים אחרות: האם מישהו לומד משהו?)
ו. נא להכין את רשימת הסטונדטים ואת רשימת הקורסים שהם לומדים (במילים אחרות, מי לומד מה?).

שאלה 6

לפניך שתי שאילתות:

```
studies(X, _). .1
studies(_, X). .2
```

א. בהסתמך על בסיס הנתונים של שאלה 5 מצא לכל אחת מהשאילתות את כל התשובות האפשריות.

ב. תרגם לעברית כל אחת מהשאילתות האלו.

שאלה 7

לפניך שלושה בסיסי נתונים:

בסיס נתונים א':

```
tree(green).
grass(green).
sky(blue).
sky(grey).
house(blue).
house(green).
house(grey).
```

בסיס נתונים ב':

```
green(tree).
green(grass).
blue(sky).
grey(sky).
blue(house).
green(house).
grey(house).
```

בסיס נתונים ג':

```
colour_of(tree, green).
colour_of(grass, green).
colour_of(sky, blue).
colour_of(sky, grey).
colour_of(house, blue).
colour_of(house, green).
colour_of(house, grey).
```

התבונן בשאלות שלהלן:

1. לאילו עצמים יש צבע ירוק?
2. מהם צבעי השמיים?
3. לאילו עצמים יש צבע כחול?
4. מה צבע העץ?
5. האם הדשא הוא ירוק?
6. האם הבית הוא אפור?

עליך לתרגם את השאלות האלו לשאילתות של פרולוג לפי:

א. בסיס נתונים א'.
ב. בסיס נתונים ב'.
ג. בסיס נתונים ג'.

איזה מבנה נתונים נראה לך יעיל והגיוני יותר: זה של בסיס הנתונים א', זה של ב' או זה של ג'? נמק את תשובתך.

5.5 שאילתה מורכבת ומשתנה משותף

לפנינו הפסוק הבא:

"אברהם הוא ההורה של יצחק ויצחק הוא ההורה של יעקב".

פסוק זה מורכב משני פסוקים אטומיים באמצעות הקשר הלוגי "ו" ("וגם", AND). את לוח האמת של הקשר הזה הגדרנו בחטיבה הראשונה. אין בפרולוג עובדות מורכבות. כלומר, עובדה של פרולוג היא תמיד פסוק אטומי. עם זאת, פרולוג מאפשר הצגת שאילתות מורכבות. הסימון התחבירי של הקשר "ו" ("וגם", AND) הוא "," (פסיק). השאילתה המורכבת של פרולוג נראית כך:

בפרולוג העברי:

```
?- הורה(אברהם, יצחק), הורה(יצחק, יעקב).
```

בפרולוג האנגלי:

```
?- parent(abraham, isaac) ,parent(isaac, jacob).
```

התשובה לשאילתה מורכבת תואמת את לוח האמת של הקשר "וגם". היא חיובית ("כן" בפרולוג העברי, yes בפרולוג האנגלי), אם ורק אם, כל אחד מהפסוקים המרכיבים את השאילתה, שייך לבסיס הנתונים. במקרה של השאילתות דלעיל, יהיה זה בסיס נתונים The Family (ראה סעיף 5.2). נציין, כי בתכנית לוגית כדוגמת The Family, סדר הפסוקים שבשאילתה אינו חשוב כלל, שהרי בחטיבה הראשונה אמרנו, כי הפסוק a AND b שקול לפסוק b AND a.

נעיין בדוגמאות נוספות:

```
?- הורה(שרה, יצחק), נקבה(שרה).
כן.
```

ניתן להרכיב שאילתות ממספר כלשהו (סופי) של פסוקים אטומיים:

?- הורה(יצחק, יעקב), זכר(יצחק), זכר(יעקב).
כן.
?- הורה(יצחק, יעקב), הורה(רבקה, יעקב), זכר(יצחק), זכר(יעקב),
נקבה(רבקה).
כן.

ניתן גם לשאול **שאילתה מורכבת עם משתנים**. תחום של המשתנה בשאילתה מורכבת הוא כל השאילתה ולכן, אם נשאל:

?- זכר(X), נקבה(X).

נקבל תשובה "לא", שהרי אין עצם בבסיס הנתונים The Family, אשר הוא גם זכר וגם נקבה. מאידך, בתשובה לשאילתה:

?- זכר(X), נקבה(Y).

נקבל את כל הזוגות שבבסיס הנתונים The Family (ללא קשר משפחתי מוגדר):

X = אברהם.
Y = שרה;

X = אברהם
Y = רבקה;

X = יצחק
Y = שרה;

וכן הלאה.

נעיין בשאילתה:

?- הורה(אברהם, X), זכר(X).

המשתנה X של השאילתה הזאת מכונה **משתנה משותף**. אם נפרש את השאילתה באמצעות שלוש הקושיות, אשר הצגנו בסעיף 5.3, תהיה הקושיה הראשונה כדלהלן:
האם קיימת הצבה תחת המשתנה X, עבורה ערכי-אמת של הפסוק הראשון **וגם** הפסוק השני, שניהם יהיו T (אמת)?

את השאילתה לעיל ניתן לתרגם לעברית כך:

מי הוא הבן של אברהם?

התשובה כידוע היא:

X = יצחק.

את השאלה:

מי היא הבת של אברהם?

ניתן להציג באמצעות השאילתה הזאת:

?- הורה(אברהם, X), נקבה(X).

על פי בסיס הנתונים The Family תהיה התשובה "לא", כלומר, לאברהם אין בנות.

נדון בתפקידו של המשתנה האנונימי בשאילתה מורכבת.
נתבונן בשאילתה:

?- הורה(אברהם,_), נקבה(_).

השאילתה הזאת מתפרשת באמצעות קושיה אחת בלבד: האם קיימות הצבות (לאו דווקא זהות), שעבורן ערך-אמת של הפסוק המורכב יהיה T (אמת). לשאילתה זו נקבל תשובה חיובית "כן", שהרי יש לנו שפע של הצבות כאלו מתוך The Family, ובתוכן: הורה(אברהם, יצחק), נקבה(רבקה). אם-כן, את השאלה:

האם לאברהם יש בת?

נציג בפרולוג כך:

?- הורה(אברהם, X), נקבה(X).

להלן מספר שאלות, שאותן נציג כשאילתות מורכבות של פרולוג (הן בפרולוג העברי והן בפרולוג האנגלי).

1. מי הם ההורים של יצחק?

?- הורה(X, יצחק), הורה(Y, יצחק), זכר(X), נקבה(Y).
X = אברהם.
Y = שרה.

```
?- parent(X, isaac), parent(Y, isaac), male(X), female(Y).
X = abraham
Y = sarah
```

2. האם ליעקב יש אם?

```
?- הורה(X, יעקב), נקבה(X).
X = רבקה.
```

```
?- parent(X, jacob), female(X).
X = rivkah
```

לסיום נציג את השאילתה הבאה:

```
?- הורה(_, יעקב), נקבה(_).
?- parent(_, jacob), female(_).
```

התשובה לשאילתה זו חיובית ואף-על-פי-כן, לא תהיה זו התשובה לשאלה 2: "האם ליעקב יש אם?". נסיק זאת מתוך השיקולים אותם השמענו בדיון בשאילתה: ?-הורה(אברהם,_), נקבה(_).

5.6 הצגה של שאילתה מורכבת – תרגול

פרק זה מיועד לעבודה עצמית. נושא העבודה – שאילתה מורכבת ומשתנה משותף.
הכן עט ונייר. אין צורך במחשב.
עבודה נעימה!

שאלה 1

התבונן בבסיס הנתונים "Dictionary" (סעיף 5.3), וענה על השאילתות הבאות:

א. `?- means(father, X), begins(X, Y).`
ב. `?- means(X, even), begins(even, Y).`
ג. `?- means(X, Y), begins(Y, bet).`

ד. `?- means(father, _), begins(_, X).`
ה. `?- means(_, even), begins(even, Y).`
ו. `?- means(X, _), begins(_, bet).`

שאלה 2

התבונן בבסיס הנתונים Dictionary ותרגם את השאלות הבאות לשאילתות של פרולוג:

א. מהו התרגום לעברית של המילה pilot, ומהי האות הראשונה של המילה המתורגמת?
ב. מהי האות הראשונה של המילה העברית zvuv ומהו תרגומה לאנגלית?
ג. מהי המילה העברית המתחילה באות dalet ומהו תרגומה לאנגלית?

שאלה 3

התבונן בבסיס הנתונים הבא:

```
studies(dani, prolog).
studies(rafi, prolog).
studies(nili, history).
studies(rafi, history).
studies(dani, mathematics).
studies(nili, mathematics).
```

תרגם לשפת פרולוג את השאלות האלה:

א. האם מישהו לומד פרולוג וגם מתמטיקה?
ב. מי לומד פרולוג וגם היסטוריה?
ג. האם נילי ורפי נפגשים בהרצאה כלשהי?
ד. באיזו הרצאה נפגשים נילי ודני?

שאלה 4

לפניך עובדות:

```
teaches(tim, jack).
teaches(ann, liz).
teacher(tim).
teacher(liz).
```

בהסתמך על העובדות האלו, מצא את כל התשובות האפשריות לשאילתות האלה:

א. `?- teaches(X, Y), teacher(X).`
ב. `?- teaches(X, Y), teacher(Y).`
ג. `?- teaches(X, Y), teacher(_).`

האם ניתן לקבוע מי המורה ומי התלמיד על פי היחס teaches בלבד וללא היחס teacher? נמק את תשובתך.

שאלה 5

התבונן בבסיס הנתונים The Family (סעיף 5.2) וענה על השאילתות הבאות:

א. `?- parent(X, Y), male(Y).`
ב. `?- parent(X, Y), male(X), female(Y).`

תרגם את השאילתות א׳ ו-ב׳ של פרולוג לשאלות בשפה העברית.

שאלה 6

התבונן בבסיסי הנתונים א׳, ב׳, ג׳, של שאלה 6 שבסעיף 5.4 וקרא את השאלות שלהלן:
א. האם קיים עצם כלשהו, אשר צבעיו אפור וכחול?
ב. האם העץ, הדשא והבית הם בעלי אותו צבע?
ג. מהו שם העצם, אשר צבעיו אפור וירוק?
ד. מה הם העצמים, אשר הצבעים שלהם הם כצבעי השמיים?

הצג כל אחת מהשאלות שלעיל כשאילתות של פרולוג על פי:
א. בסיס נתונים א׳.
ב. בסיס נתונים ב׳.
ג. בסיס נתונים ג׳.

ציין אילו מן השאלות לא ניתן היה לתרגם לשאילתות של פרולוג לפי אחד מבסיסי הנתונים.

5.7 עובדות עם משתנים

כאשר עובדה מסוימת נכונה לגבי קבוצה גדולה של עצמים, אשר את שמותיהם אין אנו רוצים לפרט, אנו נעזרים בעובדות עם משתנים. נעיין למשל בסיפור הבא:

כל אחד הוא בן-אדם.
כל אחד הוא בן-תמותה.
כל אחד אוהב את החיים.

הטענות שבסיפור דלעיל נכונות עבור כל אחד. נעיין בעובדות שלהלן:

```
בן_אדם(X).
בן_תמותה(X).
אוהב(X, חיים).
```

בהסתמך על העובדות האלו, התשובה לשאילתה הבאה היא חיובית:

```
?- בן_אדם(משה).
```

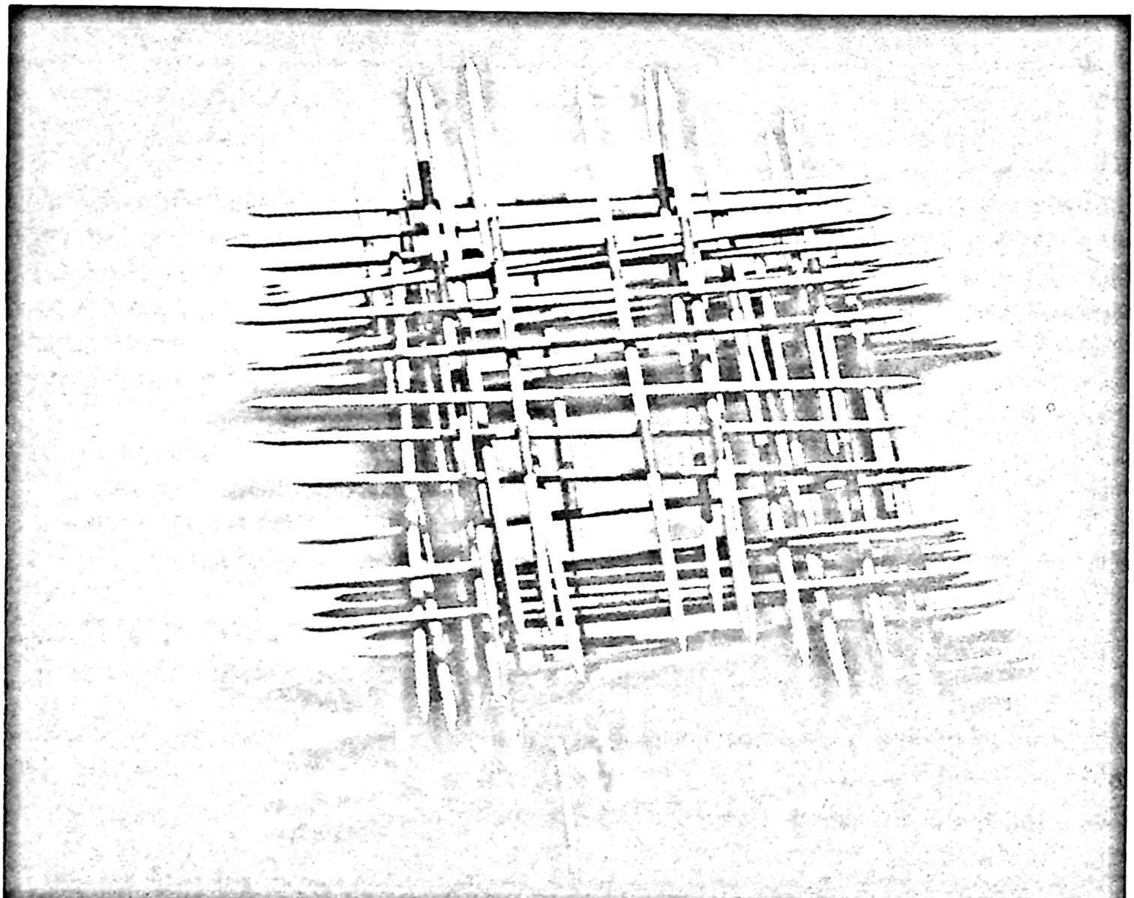
כמו-כן, חיובית גם התשובה לשאילתה זו:

```
?- בן_תמותה(כדור_הארץ).
```

וגם לשאילתה הזו התשובה היא חיובית:

```
?- אוהב(גלגל, חיים).
```

אין לפרולוג ידע שגור ואין הבחנה של הכוונות הנסתרות של כותב התכנית. כאשר אנחנו אומרים "כל אחד הוא בן-אדם" אנו מתכוונים לכל אחד מאיתנו, כלומר אל בני-אנוש. מאידך, העובדה כי לכל X בן_אדם(X), מתפרשת על ידי פרולוג כך: לכל X, אשר על פי כללי התחביר של פרולוג יכול להתמצות בתוך הסוגריים, מתקיים: בן_אדם(X). המסקנה היא, כי יש להיזהר מעובדות עם משתנים, אם כי אנו נזקקים להם לעיתים קרובות, כפי שנראה בהמשך.



פרק 6

השפה של לוגיקת היחסים

בפרק זה נמשיך להכיר את התשתית הלוגית שעליה מבוססת שפת הפרולוג. בהקדמה ציינו, כי מקורה של שפת הפרולוג בלוגיקת היחסים. בחטיבה הראשונה עסקנו בלוגיקה פסוקית, אשר במסגרתה חקרנו את תהליך החשיבה באמצעות מערכת היסק. את הפסוק האטומי של לוגיקה פסוקית הצגנו על כל חלקיו באמצעות אות אחת. בלוגיקת היחסים לעומת זאת, ניתן לייצג גם חלקים של פסוק אטומי, אשר אינם פסוקים בעצמם. לכן תאור העולם באמצעות לוגיקת היחסים הוא יותר עשיר ומגוון.

בשני נושאים עיקריים נדון בפרק זה:

- כמתים,
- הצגת ידע באמצעות לוגיקת היחסים.

6.1 סימני השפה וביטוייה

תחילה נכיר כמה מונחים של לוגיקת היחסים, המשמשים את שפת הפרולוג לתאור בעיות ולייצוג מידע:

- **פרדיקטים.**
  באמצעות הפרדיקטים מתארים את היחסים בין העצמים. הפרדיקט יכול להיות חד-מקומי, דו-מקומי, או רב-מקומי, בהתאם למספר העצמים, שאת היחס ביניהם הוא מתאר.

- **קבועים.**
  הקבועים משמשים לתאור העצמים.

- **משתנים.**
  במשתנים לוגיים נעזרים כדי לסמן ערכים שאינם ידועים.

■ **קשרים.**
אלה הם הקשרים: NOT, AND, OR, IF-THEN, IF-AND-ONLY-IF. בחטיבה הראשונה הצגנו אותם והגדרנו את לוחות האמת שלהם.

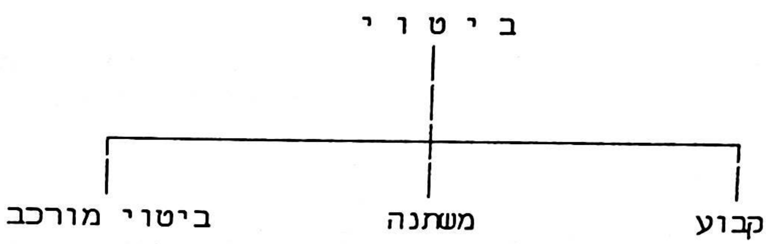
■ **כמתים** (כ׳ פתוחה, מ׳ קמוצה).
אלה הם שני סימנים, שאת משמעותם נכיר בהמשך. הסימן ∀ הוא אות A הפוכה (האות הראשונה של המילה האנגלית all). הסימן ∃ הוא אות E הפוכה (האות הראשונה של המילה האנגלית exists, שפירושה: יש, קיים). הסימן ∀ מציין **כמת כללי** (universal quantifier), והסימן ∃ מציין **כמת קיומי** (existential quantifier). נעיר כי לעיתים משתמשים במונחים: **כולל** במקום כללי, ו-**יישי** במקום קיומי.

■ **סימני פיסוק.**
הסוגרים והפסיק הינם סימני פיסוק. נשתמש בסוגריים עגולים ומרובעים. הסוגרים העגולים מציינים את סדר הקדימויות של הקשרים ומקיפים את העצמים של הפרדיקט. הפסיק מפריד בין העצמים. בסוגריים מרובעים ניעזר בכתיבת הכמתים.

את הפרדיקטים, הקבועים והמשתנים נכתוב בתעתיק הלועזי. את הפרדיקט נסמן באות קטנה או בסדרת אותיות ומספרים, אשר מתחילה באות קטנה ואיננה כוללת רווחים. סדרה זו קרויה **אטום**. הפרדיקט הוא **אטום**. הקבוע הוא אטום, או **מספר**. את המשתנה נסמן באות גדולה, או בסדרה של אותיות וספרות ללא רווחים המתחילה באות גדולה. להלן מספר דוגמאות:

משתנה	אטום
Parent	parent
X	x
Data1	data1

את העולם אשר סביבנו, מתארים בלוגיקת היחסים באמצעות ביטויים. הגדרת הביטוי היא **רקורסיבית** (משמע – ביטוי מוגדר באמצעות עצמו). **הביטוי** (term) הוא קבוע, או משתנה, או ביטוי מורכב. ביטוי מורכב בנוי מפרדיקט וארגומנטים. הארגומנטים מוקפים בסוגריים ומופרדים בפסיקים. כל אחד מהארגומנטים הוא גם-כן ביטוי, כלומר קבוע או משתנה או ביטוי מורכב. הפרדיקט הינו אטום בלבד.



איור 6.1 – הגדרה של ביטוי

להלן דוגמאות של ביטויים מורכבים:

```
parent(abraham, isaac).                        .א
couple(father(abraham), mother(sarah)).        .ב
couple(Man, Woman).                            .ג
```

הארגומנטים של ביטוי ב': father(abraham) ו-mother(sarah), הם ביטויים מורכבים. הארגומנטים של ביטוי ג': Man ו-Woman, הם משתנים. רצוי לסמן משתנה במילה שלימה ולא באות יחידה, כדי לציין את מהות העצם הלא-מפורש ולהצביע על סדר העצמים בקבוצה הסדורה: במקום הראשון – הגבר של הזוג ובמקום השני – האישה.
נעיין בדוגמאות נוספות:

```
likes(tom, sisters(ann, mary)).                      .ד
likes(Somebody, sisters(Sister1, Sister2)).          .ה
```

הארגומנט הראשון של ביטוי ד' הוא אטום, והארגומנט השני הוא ביטוי מורכב: sisters(ann, mary). הארגומנט הראשון של ביטוי ה' הוא משתנה והארגומנט השני הוא ביטוי מורכב: sisters(Sister1, Sister2). המשתנים Sister1 ו-Sister2 אינם זהים זה לזה מבחינה תחבירית.
נציין, כי את הפרדיקט, שמהווה חלק של ארגומנט (כמו sisters בדוגמה ה', או father ו-mother בדוגמה ב') מכנים בשם "פונקציה".
נציין גם, כי פרקים של לוגיקת היחסים הרלוונטיים לנושא של פרולוג שייכים ללוגיקה מסדר ראשון. משמע, שקיימת לוגיקת יחסים מסדר יותר גבוה, אשר לא נעסוק בה במסגרת ספר זה. קוראים, אשר ירצו להכיר את הנושא של לוגיקה מתמטית, יפנו לאחד מספרי הלימוד הרבים כמו למשל, לספר [Church, 1956].

6.2 השימוש בכמתים

האטום והביטוי הם מונחים השייכים לתחביר השפה. עתה נדון בסמנטיקה, שהיא – המשמעות של הנאמר. נעיין בביטוי:

```
couple(father(abraham), mother(sarah)).
```

ביטוי זה מבטא את העובדה, כי האב אברהם והאם שרה קשורים באמצעות היחס couple (זוג). ניתן לבדוק אם העובדה הזו נכונה, אם לאו, ולכן ניתן להחליט אם לעובדה זו יש משמעות של אמת (כלומר, ערך-אמת T), או משמעות של שקר (כלומר, ערך-אמת F). לעומת זאת, קשה

לדעת אם ערך האמת של הביטוי couple(Man, Woman) הוא T או F. כל עוד אין רמז באשר למה שמבטאים באמצעות המשתנים Man ו-Woman.

יתכן, שבאמצעות couple(Man, Woman) רצינו לומר, כי:

1) Every man and every woman are a couple.

המשפט הלשוני הזה הינו פסוק. הוא זכאי לתואר T (אמת) או F (שקר) ויש להניח כי ערך האמת שלו הוא F (שקר). אך יתכן, שהכוונה היתה כי:

2) Every man has a woman.

גם זה פסוק, וערך האמת שלו יכול להיות שונה מזה של הפסוק הראשון. נבחן פסוק נוסף:

3) There is a man and there is a woman, which are a couple.

פסוק זה משתמע גם הוא מהביטוי couple(Man, Woman). ראינו אם כן, כי הביטוי couple(Man, Woman) איננו חד-משמעי. כדי לציין איזה מתוך שלושת הפסוקים שלמעלה מבטא הביטוי couple(Man, Woman), עלינו להיעזר בכמתים. משמעות הכמת הכללי ∀, כאשר הוא נכתב לפני המשתנה X הינה: **for all X** (לכל X). משמעות הכמת הקיומי ∃, הנכתב לפני המשתנה X, הינה: **there exists X** (קיים X).

נציג את הפסוק הראשון באמצעות הביטוי והכמתים. את "Every man" נתרגם כך: ∀Man, ואת "every woman" נתרגם כך: ∀Woman. בתעתיק הלוגי נציג את הפסוק הראשון כך:

1) ∀Man∀Woman[couple(Man, Woman)].

נציג את הפסוק השני באמצעות הביטוי והכמתים:

2) ∀Man∃Woman[couple(Man, Woman)].

את ∃Woman מתרגמים לעברית כך: יש אישה, או קיימת אישה. הפסוק השני הוא כאמור: לכל איש קיימת אישה, אשר ביחד הם זוג.
נציג את הפסוק השלישי בעזרת הביטוי והכמתים:

3) ∃Man∃Woman[couple(Man, Woman)].

להלן דוגמאות נוספות של הצגת עובדות באמצעות ביטויים וכמתים:

4) Every child has a parent.

∀Child∃Parent[has(Child, Parent)].

המשמעות היא: לכל ילד קיים הורה, אשר הוא ההורה שלו. הילד וההורה אינם מוגדרים באופן מפורש ולכן הם מיוצגים באמצעות משתנים. המילים "לכל" ו-"קיים" מיוצגות על ידי הכמתים בהתאמה.

5) Tom likes every lady.

$\forall$Lady[likes(tom, Lady)].

6) But there is only one lady whom Tom loves.

$\exists$Lady[loves(tom, Lady)].

הבחור טום צויין במפורש, אך לא כך הגברות. לכן, המילה "טום" מיוצגת באמצעות קבוע, והמילה "גברת" – על ידי משתנה.

6.3 כמתים לא תקניים ומגבלות של ייצוג ידע באמצעות לוגיקת היחסים

נדון בהיבטים נוספים של הצגת ידע בעזרת הכמתים. לוגיקת היחסים משתמשת בשני כמתים בלבד: הכמת הכללי $\forall$ והכמת הקיומי $\exists$. כאשר ידוע כי המידע נכון עבור **כל עצם** מתחום הגדרת המשתנה, נעזרים בכמת הכללי $\forall$. כשיודעים שקיים **לפחות עצם אחד** עבורו המידע נכון, משתמשים בכמת הקיומי $\exists$. מכאן, שמשתמשים בכמת $\exists$ גם כשהמידע נכון עבור עצם אחד בלבד, וגם כאשר המידע איננו נכון עבור עצם אחד מתוך רבים.
בלשון חופשית ניתן לומר, כי הכמתים של לוגיקת היחסים דנים בשאלת הכמות. בטרם נולד המושג של מספרים טבעיים, השתמש האדם בשני מונחים בלבד להגדרת הכמות: אחד והרבה. מאז עברנו דרך ארוכה ובנוסף להגדרות מספריות של כמות, פיתחנו שורה של מונחים המבטאים כמויות שונות. להלן רשימה חלקית של כמתים, המכונים **לא-תקניים** [Mamdani & Efstathiou, 1985].

each, every, any, all, much, most,
many, lots of, plenty, some, a few,
several, hardly one, almost none.

על אף השימוש בהם בשפה הטבעית, הכמתים האלה אינם מיוצגים בלוגיקת היחסים.

נעיין בפסוק:

1) Men are brave.

נציג אותו באמצעות סמלים לוגיים:

$\forall$Man[brave(Man)].

משמעות הביטוי brave(Man) עם הכמת הכללי $\forall$ ליד המשתנה Man היא, שכל גבר הוא אמיץ. נניח כי תחום הגדרת המשתנה Man הוא כל הגברים שבעולם. האם כל גבר אמיץ? כנראה שלא, אם כי יש להניח, שרוב הגברים הם אכן אמיצים. לכן, נכון היה לומר:

2) Most men are brave.

או:

3) Some men are brave, and some are not.

איך נתרגם לשפת לוגיקת היחסים את הפסוקים 2 ו-3? איך נבטא באמצעות הסמלים של

לוגיקת היחסים את ההבדל העדין שביניהם? לצערנו, אין לנו מרשם לכך. לפיכך יש קושי, אשר נתקלים בו לעתים בבניית מאגרי מידע. נעיין בפסוקים נוספים:

4) Probably all men are brave.
5) Almost all men are brave.
6) Almost always men are brave.
7) Lots of men are brave.

ללוגיקת היחסים אין מנגנון לטפל בדקויות אלו. לכן כשאנו מתרגמים את הפסוקים 7-2 לשפה הפורמלית של לוגיקת היחסים, אנו מאבדים מעושרם המילולי, שהרי אם קיים ספק לגבי אומץ לב של גבר אחד בלבד, אנו נאלצים להשתמש בכמת הקיומי ∃ ולא בכמת הכללי ∀.

לא נרחיב את הדיון בנושא הכמתים במסגרת ספר זה. נציין רק, כי לטיפול בכמתים לא-תקניים יש חשיבות רבה בתחומים שונים של בינה מלאכותית, ובפרט בתחומי הבנה של שפות טבעיות על-ידי המחשב, או הצגת ידע במערכת מומחה.

נעיר עוד, כי בנוסף ללוגיקות מסורתיות (לוגיקה פסוקית, לוגיקת היחסים), פותחו לוגיקות חדישות, כגון: modal logic [Hughes & Cresswall, 1968], fuzzy logic [Zadeh, 1973] ואחרות. בעזרתן ניתן לטפל בכמתים הלא-תקניים ביתר גמישות. נעשים גם נסיונות לשלב לוגיקות חדישות בתוך מערכת של פרולוג [Ishizuka & Kanai, 1985].

6.4 נוסחה ופסוק בלוגיקת היחסים

פסוק אטומי שימש כיחידת שפה של לוגיקה פסוקית, אשר לא ניתן היה לפרקה ליחידות קטנות יותר. נתבונן בביטוי של לוגיקת היחסים. ביטוי זה מבטא יחס בין עצמים. הוא מורכב מפרדיקט שמבטא את היחס ומארגומנטים שמתארים את העצמים. גם אותו אי אפשר לפרק מבלי לשנות את הנאמר בו, ובשל כך מכונה הביטוי בשם **נוסחה אטומית** (atomic formula).

נוסחאות אטומיות משמשות כאבני הבנין, אשר מהן בונים את הנוסחאות של לוגיקת היחסים. כמלט בבניה זו משמשים הקשרים, הסוגריים והכמתים. מכאן שנוסחה של לוגיקת היחסים הינה נוסחה אטומית, או אוסף של נוסחאות אטומיות, הקשורות באמצעות הקשרים, והמוקפות בסוגריים ובכמתים. נבטא את מושג הנוסחה באמצעות הגדרה רקורסיבית.

נוסחה היא:

★ נוסחה אטומית.
דוגמה: $p(X)$
★ שלילתה של נוסחה אטומית.
דוגמה: $\neg p(X)$
★ אוסף של נוסחאות המחוברות באמצעות הקשרים.
דוגמה: $p(X) \vee q(X)$
★ אוסף של נוסחאות המוקפות בכמתים.
דוגמה: $\exists X[p(X) \vee q(X)]$

את הנוסחאות של לוגיקת היחסים מכנים בלע"ז well-formed formulas ומסמנים בראשי התיבות WFFS.
הנוסחאות של לוגיקת היחסים משמשות להצגת מידע. נביא כמה דוגמאות:

1) John is American.

american(john).

2) John has a big car.

has(john, big_car).

3) John has everything.

∀X[has(john; X)].

4) Americans have everything.

∀X∀Y[american(X)→has(X, Y)].

5) There are poor americans also.

∃X[poor(X)∧american(X)].

6) There are things that poor people don't have.

∀X∃Y[poor(X)→(¬has(X, Y))].

או:

∀X[poor(X)→¬(∀Y[has(X, Y)])].

כאשר לפנינו נוסחה עם כמתים, אנו אומרים כי הסוגריים המרובעים מסמנים את **טווח הכימות של המשתנה** (או המשתנים). מבחינה פורמלית, המשתנים יכולים להופיע בנוסחה ללא כימות, כמו למשל:

(poor(X)∨have(X,Y))→unhappy(X).

אותו משתנה יכול להופיע בנוסחה גם בטווח הכימות שלו וגם מחוץ לו, כמו למשל:

(∃X[poor(X)]∨have(X,Y))→unhappy(X).

המשתנה של הנוסחה נקרא **משתנה מכומת** (bound variable), אם הוא נמצא בטווח הכימות שלו, והוא נקרא **משתנה חופשי** (free variable), אם הוא נמצא מחוץ לטווח הכימות. המשתנה X בדוגמה האחרונה הוא גם מכומת וגם חופשי.
בסעיף 6.2 כשדנו במשמעות הנוסחה couple(Man, Woman) ציינו, כי לא ניתן ליחס משמעות של אמת, או משמעות של שקר לנוסחה שמכילה משתנים חופשיים. לעומת זאת, ניתן לאמת נוסחה שמשתניה מכומתים, או שכלל אין לה משתנים. על-כן, נוסחה שאין בה משתנים חופשיים הינה **פסוק** והיא זכאית לתואר אמת או שקר. נציין, כי משתני התכנות הלוגי הם משתנים מכומתים, אף-על-פי שבתכנית לוגית לא מציינים זאת במפורש.
אם כן, פסוק הוא נוסחה ללא משתנים, או נוסחה בה כל אחד מן המשתנים הוא משתנה מכומת. נוסחה אטומית ללא משתנים מקבלת ערך-אמת T (אמת) או F (שקר), בהתאם לנאמר

בה. למשל, ערך-אמת של הנוסחה likes(tom, mary) הוא T, אם טום אוהב את מרי, והוא F אם טום אינו אוהב את מרי. את ערך האמת של פסוק ללא משתנים מחשבים על פי לוחות-אמת של הקשרים ומתוך ערכי-אמת של נוסחאות אטומיות, מהן הוא מורכב.

באשר לערך-אמת של פסוק עם משתנה: הכמת הקיומי בפסוק ∃X[p(X)] מצביע על כך, שקיים לפחות X אחד, עבורו יש ל-p(X) ערך-אמת T. נעיין בפסוק אחר: ∀X[p(X)]. הכמת הכללי בו מציין, כי ערך-אמת של p(X) הוא T עבור כל X.
המשתנים של פרולוג הם משתנים מכומתים ועל-כן, העובדות והשאילתות של תכנית לוגית הם פסוקים. המשתנים של העובדה מכומתים כללי, ובשל כך לפסוק של פרולוג **בן-אדם**(X) יש ערך-אמת T גם עבור X=**משה**, וגם עבור X=**שולחן**, כפי שציינו בסעיף 5.7. משתני השאילתה מכומתים קיומי, ואת זה ביטאנו על ידי הצגת קושיות בסעיף 5.3. הצגת השאילתה: **?-זכר**(X), דומה להצגת השאלה: האם קיים X שעבורו מתקיים היחס **זכר**.

6.5 שקילות, עקביות, נביעה וכללי היסק בלוגיקת היחסים

הפסוקים של לוגיקת היחסים מקיימים את תכונות השקילות, אשר הוגדרו בחטיבה הראשונה. הפסוק: poor(john)→american(john) שקול, למשל לפסוק: ¬poor(john)∨american(john)

$$poor(john) \rightarrow american(john) \Leftrightarrow \neg poor(john) \vee american(john)$$

באופן פורמלי, תכונות השקילות מתקיימות גם עבור הנוסחאות:

$$poor(X) \rightarrow american(X) \Leftrightarrow \neg poor(X) \vee american(X)$$

פירוש הדבר, כי לערכי X, שעבורם מוגדר ערך האמת של poor(X) ושל american(X), יהיו הפסוקים משני צידי הסימן ⇔ בעלי ערך-אמת זהה. בנוסף מתקיימות גם תכונות שקילות אלו:

.1 $\forall X[p(X)] \Leftrightarrow \forall Y[p(Y)]$

ניתן לפרש את תכונת שקילות 1 כך : היחס p מתקיים לכל עצם, ואין זה חשוב אם מסמנים את העצם ב-X או ב-Y.

.2 $\exists X[p(X)] \Leftrightarrow \exists Y[p(Y)]$

ניתן לפרש את תכונת שקילות 2 כך: היחס p מתקיים עבור עצם אחד לפחות, ואים זה משנה איך מסמנים את העצם הזה.
פירוש תכונות 1 ו-2 הוא, שניתן לשנות את סימון המשתנה.

.3 $\neg(\forall X[p(X)]) \Leftrightarrow \exists X[\neg p(X)]$

את האגף השמאלי של תכונת שקילות 3 יש להבין כך: היחס p לא מתקיים עבור כל X. פירוש הדבר הוא, שקיים X עבורו היחס p איננו מתקיים, וזה בדיוק מה שנאמר באגף הימני.

להלן דוגמה לשימוש בתכונת שקילות 3. נעיין בפסוק:

Not everyone is happy

את החלק:

everyone is happy

של הפסוק הזה רושמים בשפת לוגיקת היחסים כך:

$\forall X[happy(X)]$

הפסוק עצמו נראה כך:

$\neg(\forall X[happy(X)])$

על פי תכונת שקילות 3, הוא שקול לפסוק

$\exists X[\neg happy(X)]$

נתרגם את הפסוק האחרון לאנגלית:

There exist someone which is unhappy

.4 $\neg(\exists X[p(X)] \Leftrightarrow \forall X[\neg p(X)]$

בעזרת שיקולים הדומים לאלו שלעיל, ניתן להראות, כי הפסוק

There does not exist someone which is happy

שקול לפסוק

No one is happy

.5 $\forall X[p(X) \wedge q(X)] \Leftrightarrow \forall X[p(X)] \wedge \forall Y[q(Y)]$

כלומר, את הפסוק:

Everyone is happy and smiling

ניתן להציג כך בשפת לוגיקת היחסים:

$\forall X[happy(X) \wedge smiling(X)]$

או כך:

$\forall X[happy(X)] \wedge \forall Y[smiling(Y)]$

את הפסוק האחרון נתרגם לאנגלית כך:

Everyone is happy and everyone is smiling.

.6 $\exists X[p(X) \vee q(X)] \Leftrightarrow \exists X[p(X)] \vee \exists Y[q(Y)]$

כלומר, הפסוק:

Someone is happy or smiling.

שקול לפסוק:

Someone is happy or someone is smiling.

בדומה לשקילות, ניתן להעביר ללוגיקת היחסים גם את התכונות "נובע-גורר" (אשר נסמן: $\Rightarrow$) ו"עקביות", שאותן הגדרנו בחטיבה הראשונה.

גם טכניקות של הוכחה על פי כללי היסק ובדרך השלילה, אשר דנו בהן בחטיבה הראשונה קבילות בלוגיקת היחסים. נדגים זאת בעזרת משל זה:

קבוצת הפסוקים 4, 5, 6 שבסעיף הקודם איננה עקבית. למה? כי פסוק 4 קובע שקיים X, שהוא גם עני וגם אמריקאי. נניח כי שמו של ה-X הזה הינו טום.
לכן, לפסוק

$poor(tom) \wedge american(tom)$

יש ערך-אמת T. המשמעות היא, כי ערכי-אמת של poor(tom) ושל american(tom) הם T. בשל היותו עני ועל פי פסוק 6, חסר לטום איזה עצם שהוא המסומן ב-Y. נניח כי Y מייצג מכשיר טלוויזיה. נציג זאת כך:

$$poor(tom) \rightarrow (\neg has(tom, television))$$

אבל טום הוא גם אמריקאי, וזה סותר את הפסוק 4. על פי פסוק זה לכל X, ובכלל זה טום=X, ולכל Y ובכלל זה טלוויזיה=Y, מתקיים:

$$american(tom) \rightarrow has(tom, television)$$

לכן, על פי modus ponens, שהוא כלל היסק נאות ניתן להסיק מחד, כי:

$$\{poor(tom), poor(tom) \rightarrow \neg(has(tom, television))\} \Rightarrow \neg has(tom, television).$$

מאידך:

$$\{american(tom), american(tom) \rightarrow has(tom, television))\} \Rightarrow has(tom, television).$$

כלומר, מקבוצת הפסוקים 4, 5, 6 נובעים דבר והיפוכו, ולכן קבוצה זו איננה עקבית.

קוראים, אשר מעוניינים להמשיך בדיון בנושאים של לוגיקת היחסים בהקשר לתכנות לוגי, שהועלו בפרק זה, יכולים לפנות אל הספרים הבאים: [Manna & Waldinger, 1985]; [Winston, 1984]; [Nilsson, 1980]. בספרים האחרונים לוגיקת היחסים מתוארת כאחד הכלים להצגת ידע בבינה מלאכותית. באשר להוכחות וכללי היסק בלוגיקת היחסים, ראה את הספר [Wos & others, 1984].

תרגיל 6.1

בכל אחת מהקבוצות שלהלן יש שני פסוקים ושתי מסקנות. תרגם אותם לשפת לוגיקת היחסים ובדוק את תקפותן של כל אחת מהמסקנות (כלומר, בהנחה כי ערך האמת של כל פסוק הינו T, בדוק האם ערך האמת של כל מסקנה הינו T).

א. פסוקים: כל אדם הוא בן-תמותה.
סוקרטס הוא אדם

מסקנות: אם מישהו אינו בן-תמותה, אז הוא אינו אדם.
סוקרטס הוא בן-תמותה.

ב. פסוקים: משה הוא חייל.
רק חיילים הם אמיצים

מסקנות: מי שאינו אמיץ, אינו חייל.
משה הוא אמיץ.

ג. פסוקים: כל אחד עובד במקום עבודתו.
לא כל אחד מרוצה ממקום עבודתו.

מסקנות: ישנו אחד, אשר אינו עובד.
חיים מרוצה ממקום עבודתו.

פרק 7
חוקים

החוקים של פרולוג מתארים את הקשרים הלוגיים בין יחסים שונים. הם משמשים להגדרת יחסים חדשים על בסיס היחסים הקיימים.

את התכנית הלוגית על כל מרכיביה: עובדה, חוק ושאילתה, ניתן להציג כמערכת היסק. את התהליך של קבלת תשובה לשאילתה ניתן להציג כתהליך הסקת מסקנות בדרך לוגית. ניתן לעקוב אחרי התהליך הזה באמצעות פרידקט המערכת trace.

הנושאים של פרק זה הם:
- הגדרה ומבנה של חוק,
- קבלת תשובות על בסיס החוקים כתהליך לוגי של הסקת מסקנות,
- מבט על מערכת הפרולוג באמצעות ה-trace,
- חוקים רקורסיביים.

7.1 הגדרת יחס באמצעות חוק

עיון מחודש בתכנית "The Family" (ראה סעיף 5.2) מגלה, כי היא כוללת מידע נוסף. מהעובדות: הורה(אברהם, יצחק) ו-זכר(אברהם) ניתן להסיק, כי אברהם הוא אביו של יצחק. אם נוסיף לכך את העובדה: הורה(יצחק, יעקב), נגלה כי אברהם הוא סבו של יעקב וכן הלאה.
אם נציג על בסיס תכנית "The Family" את השאילתה הבאה: ?-אב(אברהם, יצחק), נקבל תשובה שלילית. הסיבה לכך היא, שאין למחשב ידע שגור וכל אשר ידוע לו נמצא במאגר הנתונים בצורה מפורשת. ניתן להרחיב את המאגר ולהוסיף מידע על הקשרים המשפחתיים של בני המשפחה התנכי"ת. אנו נעשה זאת בדרך מתוחכמת: לא נוסיף עובדות חדשות, אלא נבנה חוקים שיאפשרו לפרולוג להרחיב את המעגל המשפחתי בהסתמך על העובדות הקיימות.

החוק של פרולוג מגדיר יחס חדש על בסיס היחסים הידועים. אברהם הוא האב של יצחק, כי הוא ההורה שלו והוא זכר. באופן כללי, כל הורה שהוא גם זכר, הינו אב. הפסוק האחרון הוא חוק המגדיר את היחס אב על בסיס היחסים הורה ו-זכר. נבטא את החוק הזה בתעתיק פורמלי באמצעות המשתנים והקשרים הלוגיים:

לכל X, לכל Y,
אם הורה(X,Y) וגם זכר(X) אז אב(X,Y).

לנוחותנו, נתחיל את החוק מן היחס, אשר למענו החוק הזה נבנה. את התוספת "לכל X ולכל Y" לא נכתוב במפורש.

אב(X,Y) אם הורה(X,Y) וגם זכר(X).

למדנו כי הסימן התחבירי של הקשר "וגם" הוא הפסיק: ",". הסימן התחבירי החדש אותו נלמד הוא ":-". סימן זה נועד לסמן את הקשר הלוגי "אם".

נציג אם-כן את החוק של הפרולוג העברי, המגדיר את היחס אב על בסיס היחסים הורה ו-זכר:

```
אב(X, Y) :- הורה(X, Y), זכר(X).
```

החוק מסתיים בנקודה, בדומה לעובדה.

להלן דוגמאות של חוקים נוספים, בהם היחס החדש מוגדר על בסיס היחסים של תכנית The Family:

1. הגדרה של היחס ילד:

```
לכל X, לכל Y,
X הוא הילד של Y אם Y הוא ההורה של X.
```

בתעתיק הפורמלי:

לכל X, לכל Y,
ילד(X,Y) אם הורה(Y,X).

בפרולוג:

```
ילד(X,Y):- הורה(Y,X).
```

2. הגדרה של היחס בן:

```
לכל X, לכל Y,
X הוא הבן של Y אם Y הוא ההורה של X ו-X הוא זכר.
```

בתעתיק פורמלי:

לכל X, לכל Y,
בן(X,Y) אם הורה(Y,X) וגם זכר(X).

בפרולוג:

```
בן(X,Y):- הורה(Y,X), זכר(X).
```

את היחס בן ניתן להגדיר גם בדרך אחרת. ניתן להיעזר ביחס ילד, אשר הגדרנו קודם לכן:

```
בן(X,Y):- ילד(X,Y), זכר(X).
```

3. הגדרה של היחס סב:

לכל X, לכל Y,
X הוא הסב של Y אם X הוא האב של Z ו-Z הוא ההורה של Y.

בתעתיק הפורמלי:

לכל X, לכל Y, לכל Z,
סב(X,Y) אם אב(X,Z) וגם הורה(Z,Y).

בפרולוג:

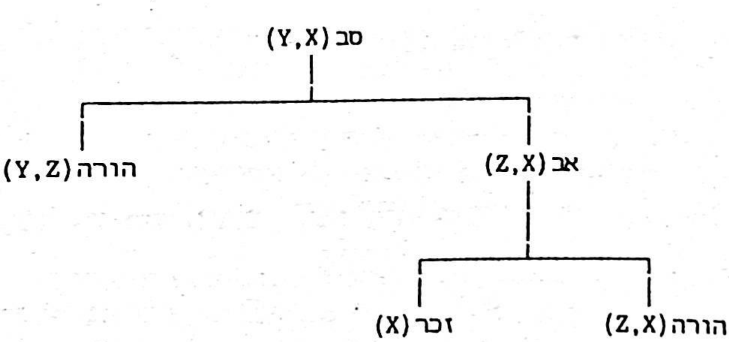
```
סב(X,Y):- אב(X,Z), הורה(Z,Y).
```

את החוקים שהגדרנו עתה ואת החוקים האחרים נאחד בתכנית לוגית, אשר אותה נכנה בשם Family Rules. התכנית הזו תתאר בעזרת חוקי פרולוג קשרי משפחה שונים ולאו דווקא קשרים של המשפחה התנכי"ת. היא תשמש כל אחד, שירצה לבנות בסיס נתונים משלו:

```
אב(X,Y):- הורה(X,Y), זכר(X).
אם(X,Y):- הורה(X,Y), נקבה(X).
סב(X,Y):- אב(X,Z), הורה(Z,Y).
סבתא(X,Y):- אם(X,Z), הורה(Z,Y).
זוג(X,Y):- אב(X,Z), אם(Y,Z).
ילד(X,Y):- הורה(Y,X).
בן(X,Y):- ילד(X,Y), זכר(X).
בת(X,Y):- ילד(X,Y), נקבה(X).
```

תכנית 7.1 – Family Rules (גירסה עברית)

נציין, כי את כל שמונת היחסים של תכנית 7.1 הגדרנו על פי שלושת היחסים האלה: הורה, זכר, נקבה. לדוגמה, את היחס סב הגדרנו על בסיס היחסים הורה ו-זכר, כפי שמציג האיור הבא:



איור 7.1 – הגדרת יחס סב על פי הורה ו-זכר

נעבור להגדרת היחסים באמצעות החוקים בפרולוג האנגלי. הסימן התחבירי החדש של הפרולוג האנגלי, שאותו נלמד, הוא ":-", ומשמעותו – IF.

```
1) For all X, for all Y,
    father(X,Y) IF parent(X,Y) AND male(X).

father(X,Y):- parent(X,Y), male(X).

2) For all X, for all Y,
    child(X,Y) IF parent(Y,X).

    child(X,Y):- parent(Y,X).

3) For all X, for all Y, for all Z,
grandfather(X,Y) IF father(X,Z) AND parent(Z,Y).

    grandfather(X,Y):- father(X,Z), parent(Z,Y).
```

נערוך את התכנית Family Rules בגירסתה האנגלית:

```
father(X,Y):- parent(X,Y), male(X).
mother(X,Y):- parent(X,Y), female(X).
grandfather(X,Y):- father(X,Z), parent(Z,Y).
grandmother(X,Y):- mother(X,Z), parent(Z,Y).
couple(X,Y):- father(X,Z), mother(Y,Z).
child(X,Y):- parent(Y,X).
son(X,Y):- child(X,Y), male(X).
daughter(X,Y):- child(X,Y), female(X).
```

תכנית 7.1 – Family Rules (גירסה אנגלית)

7.2 מבנה של חוק

אם נתעלם מן ההבדל, הקשור לכיוון הקריאה בעברית (מימין לשמאל, כנגד קריאה משמאל לימין בלע"ז), ניתן לומר כי המבנה של חוק בפרולוג האנגלי זהה לזה של הפרולוג העברי.

החוק של הפרולוג האנגלי בנוי בצורה: **a IF b**, או בתעתיק של פרולוג: **a:-b**. לאגף של החוק, אשר משמאל של ה-IF קוראים **ראש** (head), ולחלק אשר מימין קוראים **גוף** (body). אם כן, בחוק a:-b הראש הוא a והגוף הוא b. מבחינה תחבירית ניתן גם את העובדה של פרולוג לפרש כחוק, אשר גופו ריק. בראש החוק עומד ביטוי אחד בלבד, אשר מבטא את היחס שאותו החוק מגדיר. משמע, כל חוק מגדיר יחס אחד בלבד. בגוף החוק נמצאים ביטוי אחד או יותר, הקשורים באמצעות הקשרים OR, AND, NOT.

בסעיף הקודם ראינו דוגמאות של חוקים, הכוללים קשר AND. עתה נציג דוגמאות של חוקים, הכוללים גם את הקשרים OR, NOT. את הקשר OR מסמנים ברוב הגירסאות של פרולוג באמצעות נקודה-פסיק ";". נוכל להגדיר את היחס person בעזרת החוק הבא:

```
For all X,
X is a person IF
X is a male   OR
X is a female.
```

את היחס המוגדר person נבטא בפרולוג כך:

```
person(X):- male(X); female(X).
```

את היחס person ניתן גם להגדיר באמצעות שני חוקים אלטרנטיביים:

```
person(X):- male(X).
person(X):- female(X).
```

את הקשר הלוגי NOT מסמנים בפרולוג האנגלי באמצעות המילה not לפני היחס הנשלל. בפרולוג העברי מסמנים את הקשר הזה באמצעות המילה **לא**. את היחס not נלמד בהרחבה בפרק 11.

נביא דוגמה של חוק הכולל את הקשר הלוגי NOT. נגדיר את היחס unhappy באמצעות הכלל הבא:

```
For all X, for all Y,
X and Y are unhappy IF
X and Y are a couple AND
X does NOT like Y.
```

בפרולוג נבטא את הכלל הזה כך:

```
unhappy(X,Y):- couple(X,Y), not likes(X,Y).
```

את סדר קדימויות של הקשרים הלוגיים המופיעים בחוק, קבענו בפרק 1 של החטיבה הראשונה: הקדימות הראשונה היא של הקשר NOT, השניה – של AND, השלישית – של OR והרביעית – של IF. כאשר חורגים מהסדר הזה נעזרים בסוגריים. נביא דוגמה: נניח כי זוג מתגרש, אם אחד מבני הזוג אינו מחבב את השני. את המידע הזה נבטא כך:

```
For all X, for all Y,
X and Y are divorced IF
X and Y are a couple AND
       either X dislikes Y
       or Y dislikes X
```

כלומר, שני בני-אדם המסומנים ב-X ו-Y מתגרשים, אם מתקיימים שני התנאים האלה: התנאי הראשון הוא ש-X ו-Y הם אכן זוג, והתנאי השני הוא ש-X אינו מחבב את Y, או ש-Y אינו מחבב את X. את שני התנאים האלה ניתן לבטא בתעתיק הפורמלי כך:

```
couple(X,Y) AND (dislikes(X,Y) OR dislikes(Y,X)).
```

הסוגריים שמקיפים את הביטוי:

```
dislikes(X,Y) OR dislikes(Y,X)
```

מצביעים על כך, שיש כאן חריגה מן הסדר המקובל של קדימות הקשרים הלוגיים, דהיינו – הקשר OR קודם לקשר AND.

החוק הבא של פרולוג מגדיר את היחס divorsed בהתאם לתנאים שפורטו:

```
divorsed(X,Y):- couple(X,Y), (dislikes(X,Y); dislikes(Y,X)).
```

על פי תכונת הפילוג ה׳ שלמדנו בפרק 2, ניתן לבטא את תנאי הגירושין של הזוג כך:

```
couple(X,Y) AND dislikes(X,Y)  OR
        couple(X,Y) AND dislikes(Y,X).
```

בהצגה הזו של תנאי הגירושין, הקשר OR מחבר את שני החלקים האלה:

החלק הראשון: `couple(X,Y)  AND  dislikes(X,Y)`

החלק השני: `couple(X,Y)  AND  dislikes(Y,X)`

לא נזקקנו כאן לסוגריים, כי בהצגה זו אין חריגה מן הסדר המוסכם של קדימות הקשרים.

את ההצגה האחרונה של תנאי גירושין ניישם בפרולוג כך:

```
divorsed(X,Y):- couple(X,Y), dislikes(X,Y);
                couple(X,Y), dislikes(Y,X).
```

כמו כן, אפשר להגדיר את היחס divorsed באמצעות שני חוקים אלטרנטיביים:

```
divorsed(X,Y) :- couple(X,Y), dislikes(X,Y).
divorsed(X,Y) :- couple(X,Y), dislikes(Y,X).
```

מבחינה סמנטית, שלוש ההצגות של היחס divorsed שקולות זו לזו. בשל הבהירות והקריאות – עדיפה ההצגה האחרונה: באמצעות שני החוקים האלטרנטיביים. באיורים 12 ו-39 של חטיבה רביעית מוצגים שתי גירסאות של תכנית "חיפוש הדרך במבוך". האחת (39) מכילה חוק עם הקשר OR, האחרת (12) מכילה שני חוקים אלטרנטיביים.

נציין כי החוקים של פרולוג מבחינה סמנטית הינם פסוקים. משתני החוק מכומתים כללי וטווח הכימות שלהם הוא החוק כולו. אף-על-פי-כן, לפעמים מתייחסים למשתנים שמופיעים בגוף

החוק ושאינם מופיעים בראשו, כאל משתנים המכומתים קיומי.
נעיין למשל בחוק הבא:

```
couple(X,Y) :- father(X,Z), mother(Y,Z).
```
יש...

ניתן לתרגם את החוק הזה כך:

```
For all X, for all Y,
X and Y are a couple IF
there exists some Z such that
X is the father of Z AND
Y is the mother of Z.
```

כלומר, המשתנה Z, שמופיע בגוף החוק ושאיננו מופיע בראשו מתפרש כמשתנה המכומת קיומי ולא כללי.

את משתני הפרולוג האנגלי ניתן לסמן במילה המתחילה באות גדולה (בדומה למשתנים של לוגיקת היחסים). סימון זה חשוב, כי הוא משפר את נוחות הקריאה של התכנית ומבהיר את משמעות המשתנים. נעיין בדוגמה הבאה:

```
couple(Man, Woman):- father(Man, Child),
                     mother(Woman, Child).
```

בפרולוג העברי ניתן לסמן את המשתנים באמצעות אוסף סמלים, המתחיל בקו_תחתון; המילה שבאה בהמשך לקו_התחתון, מבהירה את משמעות המשתנה:

```
זוג(_גבר, _אישה):- אב(_גבר, _ילד), אם(_אישה, _ילד).
```

7.3 הסקת מסקנות על בסיס החוקים: תכנית "תלמידים"

השאילתות שהוצגו לפרולוג עד עתה, והתשובות שהתקבלו, היו על בסיס עובדות בלבד. בסעיף זה נדון בתהליך, שמתבצע על-ידי מערכת הפרולוג, כדי לקבל תשובה על בסיס החוק. לשם כך נציג וננתח מספר דוגמאות. את המודל החישובי של תכנית לוגית נבנה בפרק 10. כאן נסתפק בתאור בלתי פורמלי בלבד ונסביר (באופן בלתי פורמלי) איך פרולוג עונה על שאילתות. בסעיף זה נלמד מה הוא היסוד הלוגי לקבלת התשובה.

העובדה של פרולוג, אשר אינה כוללת משתנים, מכונה בשם **עובדה בסיסית** (ground fact). תכנית לוגית 5.1 (The Family) ותכנית לוגית 5.2 (Dictionary) נבנו אם-כן, מעובדות בסיסיות. בדומה, חוק ללא משתנים יכונה **חוק בסיסי**.

נעבור לתכנית לוגית 7.2: "תלמידים". תכנית זאת מתארת את התלמידים, את המקצועות הנילמדים ואת הכינויים להם זוכים התלמידים: סופר, איש-מחשבים, חוקר שפות טבעיות.

```
לומד(נמרוד, פרולוג).
לומד(רמי, פרולוג).
לומד(צהלה, פרולוג).
לומד(ארז, אנגלית).
לומד(יובל, פרולוג).
לומד(רמי, מתמטיקה).
לומד(הילה, מתמטיקה).
לומד(ירון, מתמטיקה).
לומד(יובל, ספרות).
לומד(נמרוד, אנגלית).
לומד(ארז, ספרות).
סופר(X):- לומד(X, ספרות).
איש_מחשבים(X):- לומד(X, מתמטיקה),
                   לומד(X, פרולוג).
חוקר_שפות_טבעיות(X):- לומד(X, פרולוג),
                        לומד(X, אנגלית),
                        לא לומד(X, מתמטיקה).
```

תכנית 7.2 – תלמידים

נענה על השאילתה הבאה, המבוססת על תכנית 4:

```
?- סופר(X).
```

לשם כך נחפש עובדה, או חוק, אשר בראשם עומד היחס סופר. מצאנו כי:

```
סופר(X):- לומד(X, ספרות).
```

המטרה החדשה, או **המטרה הנגזרת** היא:

```
?- לומד(X, ספרות).
```

עכשיו נחפש עובדה או חוק, אשר היחס לומד נמצא בראשם. העובדה הראשונה מבין עובדות האלו היא:

```
לומד(נמרוד, פרולוג).
```

נבדוק את האפשרות **להציב** תחת X את שמו של נמרוד. נפסול הצבה זו, כי נמרוד אמנם לומד פרולוג, אך פרולוג **אינו זהה** לספרות. נמשיך לחפש, עד שנגלה כי יובל לומד ספרות. הביטוי:

```
לומד(X, ספרות),
```

הוא **כללי יותר** מאשר הביטוי:

```
לומד(יובל, ספרות),
```

כי הוא כולל את המשתנה X.
על-כן נציב:

X=יובל.

הצבה זו **מאחדת** את הביטויים לומד(X, ספרות) ו-לומד(יובל, ספרות). הביטוי לומד(יובל, ספרות), הוא **המאורע** של הביטוי לומד(X, ספרות), כי הוא התקבל אחרי שהחלפנו את המשתנה X בשם של תלמיד הלומד ספרות. העובדה לומד(יובל, ספרות) היא עובדה בסיסית, מכיון שאינה כוללת משתנים. הצבת שם התלמיד תחת X מתבצעת בחוק כולו, כולל את ראשו – סופר(X).

הגענו עתה לשלב הסקת המסקנות. **האקסיומות**, או **ההנחות**, הן החוק והעובדה הבסיסיים כדלהלן:

```
סופר(יובל):- לומד(יובל, ספרות).
לומד(יובל, ספרות).
```

ובכן, החוק קובע כי יובל הוא סופר, אם יובל לומד ספרות. העובדה קובעת כי יובל אכן לומד ספרות. המסקנה היא: סופר(יובל).

את המסקנה קיבלנו על פי **כלל היסק נאות** **modus ponens** בדרך זו: נסמן את הפסוק לומד(יובל, ספרות) ב-a ואת הפסוק סופר(יובל) ב-b. בהתאם לנאמר בפרק 4 הפסוק b הוא תולדה של הפסוקים {a, a→b} על-פי modus ponens.

נציין, כי את המסקנה הנ"ל ניתן היה לנמק גם באמצעות המינוח של פרק 6 (ראה סעיף 6.5).

בתשובה לשאילתה ?-סופר(X) קיבלנו את ההצבה X=יובל, אשר פירושה הוא שהוכחנו את המשפט סופר(יובל). אם נקיש ";", נקבל X=ארז. כלומר, מהחוק המגדיר את היחס סופר ומהעובדה לומד(ארז, ספרות) נובע המשפט סופר(ארז). אם שוב נקיש ";" – נקבל את התשובה "לא".
נציג שאילתה נוספת:

```
?- איש_מחשבים(X).
```

בפניה אל תכנית 7.2 מגלים, כי X הוא איש מחשבים אם X לומד גם מתמטיקה וגם פרולוג. מאתרים את רמי, אשר לומד פרולוג ומגלים כי הוא לומד גם מתמטיקה, ומכאן מסיקים, כי X=רמי. תשובה נוספת היא: X=הילה.
נציג את השאילתה האחרונה, המבוססת על תכנית 7.2:

```
?- חוקר_שפות_טבעיות(X).
```

בפניה אל תכנית 7.2, מאתרת מערכת הפרולוג את החוק אשר ראשו חוקר_שפות_טבעיות(X). המטרה הנגזרת בהתאם לחוק זה היא לומד(X, פרולוג), אשר מצליחה בעקבות ההצבה X = נמרוד. ההצבה מתבצעת עבור כל אחד מההופעות של X בחוק הזה ולכן – המטרה הבאה היא לומד(נמרוד, אנגלית), אשר מצליחה גם היא.

כאן פונה המערכת של פרולוג אל המטרה הבאה:

```
לא לומד(נמרוד, מתמטיקה).
```

המערכת של פרולוג פועלת כאן על פי העיקרון הבא: היא בודקת אם ניתן להסיק על פי תכנית 7.2 שנמרוד אכן לומד מתמטיקה. במילים אחרות, המערכת מנסה להוכיח את ההנחה: לומד(נמרוד, מתמטיקה). היא לא מצליחה להוכיח אותה, ולכן היא מסיקה כי נמרוד אינו לומד מתמטיקה.

נסכם ונאמר כי בחיפוש אחר תשובה לשאלה, האם נמרוד לומד או לא לומד מתמטיקה, פעלה מערכת הפרולוג על פי העיקרון: "מי שלא איתנו הוא נגדנו". אם לא נכתב במפורש, שנמרוד לומד מתמטיקה, ואם לא ניתן להסיק זאת על בסיס תכנית 7.2, פירוש הדבר שאין הוא לומד מתמטיקה.
את תפקידו של קשר השלילה בתכנות לוגי נלמד ביתר הרחבה בפרק 11.

7.4 הסקת מסקנות על בסיס החוקים: תכנית Food

שם התכנית הבאה הוא Food, והיא גירסה של תכנית בעלת שם זהה המופיעה במבוא. החלטנו לאכול רק את סוגי האוכל, אשר מכילים חומרים מזינים, להם אנו זקוקים לקיומנו.

```
eaten(Food, Reason):-
      contains(Food, Part),
      needed_for(Part, Reason).
```

גם החלטנו לאכול רק את האוכל, אשר מכיל שני מרכיבים שונים לפחות:

```
eaten(Food):-
      contains(Food, Part1),
      contains(Food, Part2),
      not_same(Part1, Part2).
```

אלו הם החוקים. היחס eaten(Food,Reason) והיחס eaten(Food) הם יחסים שונים לחלוטין. שפת הפרולוג מסווגת את היחסים על פי שם הפרדיקט (הקרוי **פנקטור** לצורך זה) **וגם** על פי מספר הארגומנטים. בשל כך מסמנים את היחס הראשון eaten/2, כלומר, eaten עם שני ארגומנטים ואת היחס השני – eaten/1 כלומר, eaten עם ארגומנט אחד.

להלן התכנית Food:

```
eaten(Food, Reason):-
      contains(Food, Part), needed_for(Part, Reason).
eaten(Food):-
      contains(Food, Part1), contains(Food, Part2),
      not_same(Part1, Part2).
contains(meat, protein).
contains(meat, fats).
contains(vegetables, protein).
contains(milk, fats).
needed_for(protein, growth).
needed_for(protein, energy).
needed_for(fats, energy).
not_same(protein, fats).
```

תכנית 7.3 – Food

נציג שאילתה על פי תכנית 7.3 ונראה איך מתקבלת התשובה:

```
?- eaten(Food, Reason).
```

הפעולה הראשונה תהיה איתור החוק, אשר בראשו eaten/2. החוק הוא:

```
eaten(Food,Reason):-
        contains(Food, Part),
        needed_for(Part, Reason).
```

בהתאם לנאמר בחוק הזה, עוברת התכנית אל המטרה הנגזרת:

```
contains(Food,Part)
```

התכנית מאתרת את הפסוק הראשון, אשר ראשו הוא contains/2, ובמקרה שלנו זוהי העובדה: .

```
contains(meat, protein)
```

התכנית מבצעת את ההצבה:

```
Food = meat          Part = protein
```

ההצבה הזו מתבצעת בכל מופע של המשתנים Food ו-Part בחוק של eaten/2.
המטרה הנגזרת הבאה מתוך eaten/2 היא: needed_for(protein,Reason).
על פי העובדה needed_for(protein,growth) מתבצעת ההצבה
Reason = growth. נקבל אם כן:

```
?- eaten(Food, Reason).
Food = meat
Reason = growth
```

מבחינה לוגית הוכחנו את המשפט eaten(meat,growth) באמצעים אלה:

א. מתוך קבוצה של אקסיומות:

```
eaten(Food, Reason):-
        contains(Food, Part),
        needed_for(Part, Reason).
contains(meat, protein).
needed_for(protein, growth).
```

ב. באמצעות ההצבה:

```
Food = meat, Reason = growth, Part = protein
```

ג. על-ידי שימוש ב-modus ponens.

נבחן איך מתקבלות תשובות נוספות לשאילתה:

```
?- eaten(Food, Reason).
```

כאשר מקישים את התו ";" התכנית **נסוגה צעד לאחור**, כלומר, חוזרת אל המטרה (needed_for(protein, Reason. היא מאתרת את הפסוק (needed_for(protein,energy ומבצעת בהתאם את ההצבה Reason = energy.
התשובה הנוספת היא:

```
Food = meat
Reason = energy
```

המשפט (eaten(meat, energy הוכח כדלהלן:

א. מתוך האקסיומות האלה:

```
eaten_for(Food, Reason):-
        contains(Food, Part),
        needed_for(Part, Reason)
contains(meat, protein).
needed_for(protein, energy).
```

ב. באמצעות ההצבה:

```
Food = meat, Reason = energy, Part = protein
```

ג. על-פי modus ponens.

אנו מעוניינים בתשובות נוספות. שוב מקישים ";" ושוב חוזרת התכנית אל המטרה (needed_for(protein, Reason. היא בודקת את האפשרויות להציב שם-עצם השונה מ-energy, תחת המשתנה Reason. היא מגלה כי אין הצבות נוספות **ונסוגה צעד נוסף לאחור** אל המטרה (contains(Food, Part. היא מאתרת את העובדה (contains(vegetable,protein ומציבה Food = vegetable ו-Part = protein. התכנית עוברת אל המטרה (needed_for(protein, Reason ומתקדמת כפי שעשתה בשלב הקודם. בסיום הרצת התכנית תהיינה לפנינו התשובות הבאות, לפי סדר קבלתן:

```
?- eaten(Food, Reason).

Food = meat
Reason = growth;

Food = meat
Reason = energy;

Food = meat
Reason = energy;

Food = vegetables
Reason = growth;

Food = vegetables
Reason = energy;

Food = milk
Reason = energy;

no (more answers)
```

התשובה:Reason = energy, Food = meat, התקבלה פעמיים: בפעם הראשונה על-פי האקסיומות שצויינו לעיל, ובפעם השניה – על-פי האקסיומות הבאות:

```
eaten(Food, Reason):-
        contains(Food, Part),
        needed_for(Part, Reason).
contains(meat, fats).
needed_for(fats, energy).
```

בסעיף 7.6 נעקוב אחרי תהליך קבלת התשובות בעזרת איור 7.3.

נבדוק את תהליך קבלת התשובה לשאילתה נוספת:

```
?- eaten(Food).
Food=meat;

no(more answers).
```

קבלת התשובה Food=meat היא מבחינה לוגית כהוכחת המשפט (eaten(meat. להלן ההוכחה. אין זה אלגוריתם, אלא תאור בלתי פורמלי של מהלך התכנית.

1. **המטרה: eaten(Food).**
על-פי החוק eaten/1, פניה אל contains(Food, Part1).

2. **המטרה: contains(Food, Part1).**
על-פי הראשונה מבין העובדות contains/2, ההצבה: Food = meat ,Part1 = protein.

מצב ביניים:

```
eaten(meat):- contains(meat, protein),
              contains(meat, Part2),
              not_same(protein, Part2).
```

3. **המטרה: contains(meat, Part2).**
על-פי הראשונה מבין העובדות contains/2, ההצבה: Part2 = protein.

מצב ביניים:

```
eaten(meat):- contains(meat, protein),
              contains(meat, protein),
              not_same(protein, protein).
```

4. **נסיגה אל מצב-הביניים שלאחר השלב השני**, כי העובדה:
not_same(protein, protein) אינה שייכת לתכנית 7.3.

5. **המטרה: contains(meat, Part2).**
על-פי השניה מבין העובדות contains/2, ההצבה: Part2 = fats.

מצב ביניים:

```
eaten(meat):- contains(meat, protein),
              contains(meat, fats),
              not_same(protein, fats).
```

מצב ביניים זה הוא גם המצב הסופי.

הוכחנו את המשפט eaten(meat) באמצעות ההצבה:

```
Food = meat, Part1 = protein, Part2 = fats
```

על-פי modus ponens ומתוך האקסיומות הבאות:

```
eaten(Food):-
      contains(Food, Part1),
      contains(Food, Part2),
      not_same(Part1, Part2)
contains(meat, protein).
contains(meat, fats).
not_same(protein, fats).
```

האיור 7.4 של סעיף 7.6 מדגים, איך מערכת הפרולוג משיבה על השאילתה הנ"ל.

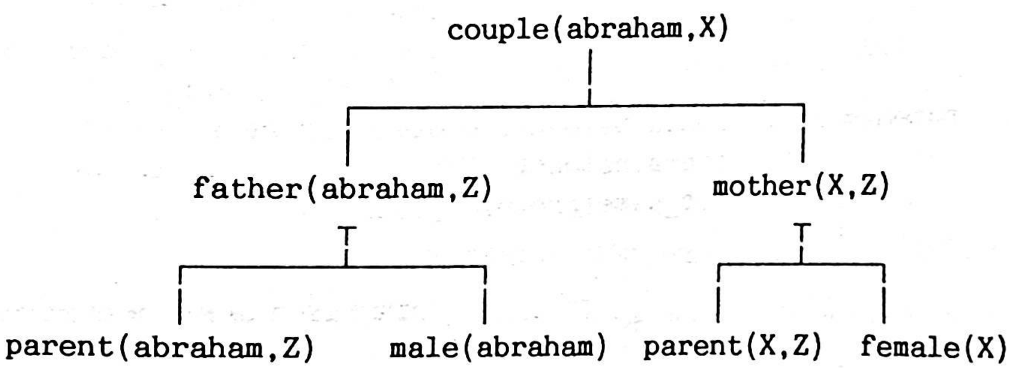
למדנו בתכנית 7.3, כי אנו אוכלים בשר וירקות כדי לגדול ולהיות נמרצים. את החלב אנו שותים רק למען המרץ. אם אין לנו זמן לאכול הרבה, כדאי לאכול בשר, כי כך אנו נהנים ממבחר יותר עשיר ומגוון של מרכיבי המזון. זוהי המשמעות הלוגית של התכנית והשאילתות אשר הוצגו.

7.5 הסקת מסקנות על בסיס החוקים: חישוב המטרה couple(abraham,X)

נביא דוגמה נוספת לתהליך קבלת התשובה. נפנה אל הגירסה האנגלית של תכניות 5.1 ו-7.1 ונבדוק מי היא בת-זוגו של אברהם-אבינו. נציג את השאילתה:

```
?- couple(abraham, X)
```

החוק שמגדיר את היחס couple, בנוי על בסיס היחסים: female, parent, male. האיור שלהלן מתאר את הקשר בין היחסים האלו לבין השאילתה שהצגנו קודם. האיור גם מתאר את המטרות הנגזרות של תהליך קבלת התשובה.
couple(abraham,X)



איור 7.2 – המטרות הנגזרות

להלן שלבי קבלת התשובה X = sarah. אין זה אלגוריתם של חישוב התשובה, אלא תאור בלתי פורמלי בלבד.

1. **המטרה:** couple(abraham, X).
על-פי couple/2, פניה אל father(abraham,Z).

2. **המטרה: (father(abraham, Z.**
על-פי father/2, פניה אל (parent(abraham,Z.

3. **המטרה: (parent(abraham,Z.**
על-פי parent/2, הצבה: Z = isaac.

מצב ביניים:

```
couple(abraham, X):- father(abraham, isaac),
                     mother(X, isaac).
father(abraham, isaac):- parent(abraham, isaac),
                         male(abraham).
mother(X,isaac):- parent(X, isaac),
                  female(X).
```

4. **המטרה: (male(abraham.**
אימות של המטרה (male(abraham ובעקבותיה –
אימות של (father(abraham, isaac.

5. **המטרה: (mother(X, isaac.**
על-פי mother/2, פניה אל (parent(X, isaac.

6 **המטרה: (parent(X, isaac.**
על-פי parent/2, הצבה: X=sarah.

7. **המטרה: (female(abraham.**
המטרה נכשלה. צעד אחד לאחור.

8. **המטרה: (parent(X, isaac.**
על-פי parent/2, הצבה: X=sarah.

9. **המטרה: (female(sarah.**
אימות (female(sarah ובעקבותיה -
אימות (mother(sarah,isaac.

התשובה היא X=sarah, שמשמעה: הוכחת המשפט (couple(abraham, sarah על-פי האקסיומות האלו:

```
1) for all X, Y, Z,
   couple(X, Y):- father(X, Z), mother(Y, Z).
2) for all X, Z,
   father(X, Z):- parent(X, Z), male(X).
3) for all Y, Z,
   mother(Y, Z):- parent(Y, Z), female(Y).
4) parent(abraham, isaac).
5) parent(sarah, isaac).
6) male(abraham).
7) female(sarah).
8) father(abraham, isaac).
9) mother(sarah, isaac).
```

בתהליך ההוכחה ביצענו את ההצבות: Y = sarah, Z = isaac.
את ההוכחה קיבלנו על פי כלל ההיסק modus ponens. להלן תמצית של שלבי ההוכחה:

א. מתוך האקסיומות 6,4,2 הוכחנו את המשפט father(abraham, isaac), וצירפנו אותו אל קבוצת האקסיומות בתור אקסיומה נוספת: אקסיומה מספר 8.

ב. מתוך האקסיומות 7,5,3 הוכחנו את המשפט mother(sarah, isaac), אשר אותו צירפנו אל הקבוצה כאקסיומה מספר 9.

ג. מתוך האקסיומות 9,8,1 הוכחנו את המשפט couple(abraham, sarah).

נעיר, כי את השילוב של הצבות ו-modus ponens מכנים לפעמים בשם universal modus ponens.

7.6 בעקבותיהן של המטרות: פרדיקט המערכת trace

היחסים בהם השתמשנו עד כה באו כתוצאה מיוזמות שלנו. יחסים מסויימים תוארו באמצעות עובדות בסיסיות, כדוגמת היחסים הורה, נקבה, זכר. יחסים אחרים הוגדרו באמצעות החוקים, כמו למשל היחסים אב, סב, זוג. בנוסף ליחסים המוגדרים על ידנו, המערכת של פרולוג מכילה קבוצת יחסים המוגדרים מראש, המכונים **פרדיקטי המערכת**. בדרך כלל אלו הם היחסים שהשימוש בהם נפוץ. לרוב, פרדיקטי המערכת מנצלים את שגרות המחשב. הם מיועדים לביצוע פעולות שונות כמו ניהול קבצים, טיפול בבסיסי נתונים, פעולות החשבון, פעולות קלט-פלט וכן הלאה. ישנם הבדלים בין פרדיקטי המערכת של גירסאות פרולוג שונות.

בסעיף זה נכיר את פרדיקט המערכת trace, אשר תפקידו לעקוב אחר קבלת התשובה לשאילתה. בעת הפעלת הפרדיקט הזה, מערכת הפרולוג מתארת את תהליך קבלת התשובה. על מסך המחשב מוצגת השאילתה וסדרת המטרות הנגזרות. נעקוב אחר קבלת התשובה לשאילתה eaten(Food,Reason) על-פי התכנית Food.
נציג את השאילתה הבאה:

```
?- trace(eaten(Food, Reason)).
```

להלן הפלט של trace, המלווה בהסבר:

השורה הראשונה של הפלט:

```
[0] call: eaten(Food(80, Reason81)
```

השורה הזאת קוראת לנסות להוכיח את השאילתה, כלומר, למצוא הצבה תחת המשתנים Food ו-Reason כך, שעבורה ערך האמת של השאילתה יהיה T (אמת). מרגע זה השאילתה מכונה בשם **מטרה**. אנו נשתמש במינוח הבא:

– **המטרה הושגה**, או **הצליחה**, אם מצאנו הצבה תחת המשתנים,
– **המטרה נכשלה**, אם נקבל תשובה שלילית (no).

המספרים 80 ו-81, שרשומים ליד המשתנים, שייכים לתחום הניהולי של מערכת הפרולוג, אשר למשתמש אין בו עניין.

השורה השניה של הפלט:

```
[1] call: contains(Food183, Part185).
```

השורה הזאת מסמלת פניה אל המטרה הנגזרת על-פי החוק שמגדיר את היחס eaten(Food,Reason). המספר שבסוגריים המרובעים מציין את עומק החיפוש. בשפה חופשית ניתן לומר כי הוא מציין בכמה מטרות התרחקנו מן השאילתה.

השורה השלישית:

```
[1] exit: contains(meat, protein).
```

השורה הזאת מבשרת את השגת המטרה. היא התקבלה בעקבות פניה אל העובדה הראשונה מבין העובדות, המתארות את היחס contains.

השורה הבאה:

```
[2] call: needed_for(protein, Reason184).
```

שורה זו מסמלת פניה אל המטרה הבאה, אשר נמצאת בגוף החוק eaten(Food,Reason).

השורות שלהלן הן:

```
[2] exit: needed_for(protein, growth).
```

כלומר, המטרה הושגה.

```
[0] exit: eaten(meat, growth).
```

כלומר המטרה המקורית הושגה. המטרה המקורית הינה השאִילתה, ועל כן הצלחתה של המטרה המקורית מסמלת את גמר תהליך ההוכחה.

השורה האחרונה של ה-trace היא:

```
...in 3 reductions.
```

המספר 3 הוא מספר המטרות אשר הוכחנו תוך כדי קבלת התשובה. האחת מהן היא השאילתה והאחרות הן המטרות הנגזרות. אל המונח reduction נחזור בפרק 10, כשנדון במודל החישובי של התכנית הלוגית.

בסיום ה-trace מתקבלת התשובה:

```
' Food = meat
  Reason = growth
```

כדי לקבל trace של תשובה נוספת, מקישים נקודה-פסיק ";" אשר מתקבלת על ידי מערכת הפרולוג כהודעת כישלון.
המערכת מגיבה כך:

```
[0] redo: eaten(meat, growth).
```

בשלב זה המערכת מבצעת את **הנסיגה** (backtracking). היא חוזרת אל המטרה האחרונה שהצליחה ודורשת לבצע אותה מחדש:

```
[2] redo: needed_for(protein, growth).
```

הנסיגה מתבצעת על-ידי מערכת הפרולוג באופן אוטומטי. היא מתבצעת בעקבות כישלון המטרה, כדי לאתר דרכים אלטרנטיביות, המובילות אל הוכחת המשפט.

להלן המשך ה-trace. בשלבים שונים של ה-trace מופיעה ההודעה fail. משמעותה של הודעת fail הינה, שהניסיון להוכיח את המטרה נכשל.

```
tracing: ¦ ?- eaten(Food, Reason).
[0] call: eaten(Food80,Reason81)?
 [1] call: contains(Food183,Part185).
 [1] exit: contains(meat,protein).
 [2] call: needed_for(protein,Reason184).
 [2] exit: needed_for(protein,growth).
[0] exit: eaten(meat,growth).
... in 3 reductions.

Food = meat
Reason = growth ;
[0] redo: eaten(meat,growth).
 [2] redo: needed_for(protein,growth).
 [2] exit: needed_for(protein,energy).
[0] exit: eaten(meat,energy).
... in 3 reductions.

Food = meat
Reason = energy ;
[0] redo: eaten(meat,energy).
 [2] redo: needed_for(protein,energy).
 [2] fail: needed_for(protein,_184).
 [1] redo: contains(meat,protein).
 [1] exit: contains(meat,fats).
 [2] call: needed_for(fats,_184).
 [2] exit: needed_for(fats,energy).
[0] exit: eaten(meat,energy).
... in 3 reductions.

Food = meat
Reason = energy ;
[0] redo: eaten(meat,energy).
 [2] redo: needed_for(fats,energy).
 [2] fail: needed_for(fats,_184).
 [1] redo: contains(meat,fats).
 [1] exit: contains(vegetables,protein).
 [2] call: needed_for(protein,_184).
 [2] exit: needed_for(protein,growth).
[0] exit: eaten(vegetables,growth).
... in 3 reductions.

Food = vegetables
Reason = growth ;
[0] redo: eaten(vegetables,growth).
 [2] redo: needed_for(protein,growth).
 [2] exit: needed_for(protein,energy).
[0] exit: eaten(vegetables,energy).
... in 3 reductions.

Food = vegetables
Reason = energy ;
[0] redo: eaten(vegetables,energy).
 [2] redo: needed_for(protein,energy).
 [2] fail: needed_for(protein,_184).
 [1] redo: contains(vegetables,protein).
 [1] exit: contains(milk,fats).
 [2] call: needed_for(fats,_184).
 [2] exit: needed_for(fats,energy).
[0] exit: eaten(milk,energy).
... in 3 reductions.

Food = milk
Reason = energy
```

איור 7.3 – בעקבות המטרה
eaten(Food, Reason)

```
tracing: ¦ ?- eaten(Food).
[0] call: eaten(Food80)?
 [1] call: contains(Food179,Part1180).
 [1] exit: contains(meat,protein).
 [2] call: contains(meat,Part2181).
 [2] exit: contains(meat,protein).
 [3] call: not_same(protein,protein).
 [3] fail: not_same(protein,protein).
 [2] redo: contains(meat,protein).
 [2] exit: contains(meat,fats).
 [3] call: not_same(protein,fats).
 [3] exit: not_same(protein,fats).
[0] exit: eaten(meat).
... in 4 reductions.

Food = meat ;
[0] redo: eaten(meat).
 [3] redo: not_same(protein,fats).
 [3] fail: not_same(protein,fats).
 [2] redo: contains(meat,fats).
 [2] fail: contains(meat,_181).
 [1] redo: contains(meat,protein).
 [1] exit: contains(meat,fats).
 [2] call: contains(meat,_181).
 [2] exit: contains(meat,protein).
 [3] call: not_same(fats,protein).
 [3] fail: not_same(fats,protein).
 [2] redo: contains(meat,protein).
 [2] exit: contains(meat,fats).
 [3] call: not_same(fats,fats).
 [3] fail: not_same(fats,fats).
 [2] redo: contains(meat,fats).
 [2] fail: contains(meat,_181).
 [1] redo: contains(meat,fats).
 [1] exit: contains(vegetables,protein).
 [2] call: contains(vegetables,_181).
 [2] exit: contains(vegetables,protein).
 [3] call: not_same(protein,protein).
 [3] fail: not_same(protein,protein).
 [2] redo: contains(vegetables,protein).
 [2] fail: contains(vegetables,_181).
 [1] redo: contains(vegetables,protein).
 [1] exit: contains(milk,fats).
 [2] call: contains(milk,_181).
 [2] exit: contains(milk,fats).
 [3] call: not_same(fats,fats).
 [3] fail: not_same(fats,fats).
 [2] redo: contains(milk,fats).
 [2] fail: contains(milk,_181).
 [1] redo: contains(milk,fats).
 [1] fail: contains(_179,_180).
[0] fail: eaten(_80).

no
```

איור 7.4 – בעקבות המטרה eaten(Food)

7.7 הצגת מידע באמצעות עובדות וחוקים – תרגול

בשלב זה נציג לפניך שאלות, אשר יש להן שתי מטרות:

* האחת – לעזור לך להעריך את התקדמותך בנושא תכנות לוגי ושפת פרולוג.
* השניה – להציג בפניך אתגרים חדשים ולספק חומר למחשבה.

אינך זקוק למחשב, כדי לענות על השאלות 1 ו-2.

עבודה מהנה!

שאלה 1

הצג בשפת פרולוג את החוקים הבאים:

```
1. For all X
       X has a child if
           there is some Y such that
           X is the parent of Y.

2. For all X, Y
       X and Y form a family if
           there is some Z such that
           X is the mother of Z and
           Y is the father of Z.
3. For all X, Y
       X is the grandfather of Y if
           there is some Z such that
           X is the father of Z and
           Z is the father of Y
       or
           X is the father of Z and
           Z is the mother of Y.

4. For all X, Y
       X is the grandmother of Y if
           there is some Z such that
           X is the mother of Z and
               either Z is the father of Y
               or Z is the mother of Y.

5. For all X, Y
       X and Y are unhappy if
           X and Y live together and
               neither X likes Y
               nor Y likes X.
```

שאלה 2

לפניך סיפור, המתאר בשפת הפרולוג את שלושת החברים יורם, יוסי ומשה:

```
אוהב(יורם, לקרוא).
אוהב(יורם, לעבוד).
אוהב(יוסי, לעבוד).
אוהב(יוסי, ספורט).
אוהב(משה, ספורט).
חרוץ(X):- אוהב(X, לעבוד).
בריא(X):- אוהב(X, ספורט).
מלומד(X):- אוהב(X, לקרוא).
מאושר(X):- חרוץ(X), בריא(X).
יתקדם_יפה_בחיים(X):- חרוץ(X), מלומד(X).
עצלן(X):- בריא(X), לא חרוץ(X).
ימצא_את_מקומו_בחיים(X):- חרוץ(X); מלומד(X).
```

1. תרגם את הסיפור משפת פרולוג לשפה העברית. לנוחותך [illegible] תוכל לשנות את סדר הפסוקים.

2. ענה על השאלות הבאות:
   א. מי מבין החברים יורם, יוסי ומשה ימצא את מקומו בחיים?
   ב. מי מבין החברים יתקדם יפה בחיים?
   ג. מי מבין החברים עצלן?
   ד. מי המאושר?
   ה. האם העצלן שבהם הוא גם המאושר?

בהקשר לנושא הצגת מידע בשפת פרולוג, ראוי להביא את הדברים הבאים:
אחת ההנחות של מחקר בבינה מלאכותית היתה, שאם נתאר את עבודת המוח האנושי בשפה פורמלית ומדויקת, כדוגמת לוגיקה או פרולוג, יוכל המחשב לחקות את תהליך החשיבה האינטליגנטית. בשנת 1956 התקיים כנס בעיר דרטמות, ארה"ב (Dartmouth Conference). בכנס נשמעו תחזיות אופטימיות בדבר הצגה פורמלית של כל היבט אנושי של [illegible] אחרי מחקר של שלושים שנה ויותר, האופטימיות קצת פחתה.

בשאלות שלהלן ננסה לבדוק האם ניתן להציג בפרולוג את כל מה שנאמר בשפה טבעית. נציג בפניכם פסוקים בשפה הטבעית ונשאל, האם ואיך ניתן לבטא אותם בשפת הפרולוג. נציג גם תכניות בפרולוג ונבקש לנתחן.
רצוי לפתור את השאלות 3-6 ללא עזרת המחשב, ולאחר מכן להריץ את התכניות על מנת להשוות את התוצאות.

שאלה 3

לפניך קבוצת פסוקים:

א. משה אוהב לבלות.
ב. כולם אוהבים לבלות.
ג. טיפש מי שאינו אוהב לבלות.
ד. יורם טיפש.

1. הצג את פסוקי הקבוצה בשפת הפרולוג.

(**רמז:** העזר ביחסים אוהב, טיפש).

2. בהנחה כי ערך האמת של כל פסוק בפרולוג הוא T, בדוק האם קיבלת קבוצת פסוקים עקבית, אם לא: נמק את תשובתך.

3. ענה על השאלות הבאות:

א. האם משה טיפש?
ב. האם יורם אוהב לבלות?
ג. האם יורם טיפש?

4. הצג את השאלות שבסעיף 3 למעלה כשאילתות של פרולוג. בדוק מה תהינה התשובות והסבר אותן. במידת הצורך העזר ב-trace.

(**הערה:** אם הגירסה אשר ברשותך היא של הפרולוג האנגלי, כתוב את היחסים ואת שמות העצמים בתעתיק הלועזי).

שאלה 4

בתכנית Family Rules ניתן להגדיר את היחס אחים (brothers) באמצעות החוק הבא:

```
brothers(Child1, Child2):- male(Child1),
                           male(Child2),
                           parent(X, Child1),
                           parent(X, Child2).
```

האם החוק הזה אמנם מבטא את היחס אחים כפי שנהוג ומקובל אצל בני אדם? נמק את תשובתך.

שאלה 5

א. נתון הפסוק:

```
All men are brave
```

נתרגם אותו ללוגיקת היחסים:

```
For all X,
brave(X) IF man(X)
```

ומכאן נעבור לפרולוג:

נעיין בפסוק אחר:

```
brave(X):- man(X).
```

Only men are brave.

נתרגם אותו ללוגיקת היחסים, ונקבל:

For all X,
brave(X) IF AND ONLY IF man(X)

האם ניתן לתרגם את הפסוק הזה לפרולוג? נמק את תשובתך.

ב. לפניך שני בסיסי נתונים:

```
1) brave1(X):- man1(X).
   man1(X):- brave1(X).
   man1(tom).
```

```
2) man2(tom).
   man2(X):- brave2(X).
   brave2(X):- man2(X).
```

מצא את התשובות של פרולוג לשאילתות הבאות:

```
?- brave1(tom).
?- brave2(tom).
```

תן הסבר לתשובות.

שאלה 6

זוג העצמים העומד ביחס דו-מקומי הוא זוג סדור, שהרי אם נהפוך את סדר העצמים, עלול היחס לחדול מלהתקיים. ישנם יחסים אשר מתקיימים גם כאשר הופכים את סדר העצמים. נעיין למשל, ביחס נשואים. אם דן נשוי לרותי, אז רותי נשואה לדן ולהיפך. יחס מסוג זה נקרא **יחס סימטרי**. היחס married הוא יחס סימטרי.

For all X, for all Y,
X is married to Y IF AND ONLY IF Y is married to X.

לפניך שני בסיסי ידע (להזכירך כי אוסף עובדות וחוקים של פרולוג מכונה בשם בסיס ידע, ראה במבוא):

```
1) married1(mary, tom).
   married1(X,Y):- married1(Y, X).
```

```
2) married2(X, Y):- married2(Y, X).
   married2(mary, tom).
```

ענה על השאילתות האלו:

```
?- married1(X, Y).
?- married2(X, Y).
```

הסבר את תוצאות.

7.8 רקורסיה

בסעיף זה נדון במושג הרקורסיה. אומרים, כי ההגדרה הינה **רקורסיבית** (recursive) אם המונח מוגדר באמצעות עצמו. ההגדרות של ביטוי ונוסחה בפרק 6 היו הגדרות רקורסיביות. בתכנות לוגי יש לרקורסיה חשיבות רבה, שכן היא אחד הכלים היעילים ביותר להגדרת היחסים, אשר אותם ניתן להמשיך ללא סוף, כמו הדורות של אבותינו.

בתכנית Family Rules 7.1 הגדרנו את היחס סב. ניתן להמשיך את שרשרת האבות ולבנות את אילן היוחסין המשפחתי, כפי שמוצג כאן:

```
greatgrandfather(X, Y):-
                father(X, Link).
                grandfather(Link, Y).
ancestor(X, Y):-
                father(X, Link1).
                father(Link1, Link2).
                father(Link2, Link3).
                father(Link3, Y).
```

שתי בעיות היו עומדות לפנינו לו היינו ממשיכים בדרך זו. האחת היא אורך התכנית והשניה, המשמעותית יותר – אורך שרשרת האבות. לסבא-של-סבא-של-סבא יש גם כן סבא, אשר אותו נקפת, אם נעצור את "חוקי סבא" בשלב אחד לפניו. את שתי הבעיות האלו פותרים באמצעות **חוק רקורסיבי** (rule by recursion) כלומר, באמצעות חוק בו היחס המוגדר נמצא גם בראשו וגם בגופו. נגדיר את היחס ancestor באמצעות שני חוקים, אשר השני שבהם הוא חוק רקורסיבי:

```
ancestor(Ancestor, Person):-
                        father(Ancestor, Person).
ancestor(Ancestor, Person):-
                        father(Ancestor, Link).
                        ancestor(Link, Person).
```

החוק השני, בו היחס המוגדר ancestor מופיע בראש ובגוף ההגדרה, נקרא **חוק רקורסיבי**. ניתן לפרש אותו כהכללת חוקי "סבא-של-סבא-של...". החוק הראשון של היחס ancestor מכונה בשם **תנאי הגבול**, או **תנאי השפה** (boundary condition).

את הדרך בה החוק הרקורסיבי פועל נסביר בצורה בלתי פורמלית באמצעות תכנית 7: "שושלת היוחסין של קובי".

```
אב(אברהם, יצחק).
אב(יצחק, יעקב).
אב(יעקב, יוסף).
/* ובהמשך...*/
אב(יוסף, אבי).
אב(אבי, צחי).
אב(צחי, קובי).
אב_בשושלת(X, Y):- אב(X, Y).
אב_בשושלת(X, Y):- אב(X, L), אב_בשושלת(L, Y).
```

תכנית 7.4 – שושלת היוחסין של קובי

הערה: ברוב הגרסאות של פרולוג, הדרך להכנסת הערות לתכנית היא להקיף את ההערה משני צדדיה בקו נטוי עם כוכבים. אין המהדר (או המפרש) מתייחס אל הנכתב בין הכוכבים. ההערה /* ובהמשך...*/ נועדה להדגיש, כי שרשרת האבות של קובי היא הרבה יותר ארוכה מזו שהוצגה בתכנית.

כדי לבדוק האם קובי הוא אכן מצאצאיו של אברהם אבינו, נציג את השאילתה:
?-אב_בשושלת(אברהם, קובי).
כאשר השאילתה מוצגת, מאתרת התכנית את הראשון מבין החוקים, אשר היחס אב_בשושלת הוא ראשו. על פי הראשון מבין החוקים, אברהם הוא האב בשושלת של קובי, **אם** הוא אביו. בתום בדיקת אב/2 התכנית מגלה כי **אין זה נכון שאברהם הוא האב של קובי**. היא פונה לחוק השני של אב_בשושלת. לפי החוק הזה, אברהם הוא האב בשושלת של קובי **אם** הוא אב של מישהו והמישהו הזה הוא אב בשושלת של קובי. התכנית מגלה כי **אברהם הוא אביו של יצחק**.

כדי לבדוק האם יצחק הוא אב בשושלת של קובי, היא פונה אל הראשון מבין החוקים של אב_בשושלת. כשהיא מגלה כי **יצחק אינו אביו של קובי**, היא פונה אל החוק השני, מגלה כי **יצחק הוא אביו של יעקב**, ובודקת אם יעקב הוא אב בשושלת של קובי, וכן הלאה. את החיפוש והגילוי נדגים על ידי הצגה בלתי פורמלית של תהליך קבלת התשובה באמצעות איור 7.5.

שאילתה: ?- אב_בשושלת(אברהם, קובי).

חוק ראשון: האם אב(אברהם, קובי)? **לא.**
חוק שני: אב(אברהם, יצחק).
האם אב_בשושלת(יצחק, קובי)? (קושיה ראשונה).

חוק ראשון: **האם** אב(יצחק, קובי)? **לא.**
חוק שני: אב(יצחק, יעקב)
האם אב_בשושלת(יעקב, קובי)? (קושיה שניה).

חוק ראשון: **האם** אב(יעקב, קובי)? **לא.**
חוק שני: אב(יעקב, יוסף).
האם אב_בשושלת(יוסף, קובי)? (קושיה שלישית).

חוק ראשון: **האם** אב(יוסף, קובי)? **לא.**
חוק שני: אב(יוסף,אבי).
האם אב_בשושלת(אבי, קובי)? (קושיה רביעית).

חוק ראשון: **האם** אב(אבי, קובי)? **לא.**
חוק שני: אב(אבי, צחי).
האם אב_בשושלת(צחי, קובי)? (קושיה חמישית).

חוק ראשון: **האם** אב(צחי, קובי)? **כן.**

המסקנה: אב(צחי, קובי),
לכן אב_בשושלת(צחי, קובי), (קושיה חמישית).
לכן אב_בשושלת(אבי, קובי), (קושיה רביעית).
לכן אב בשושלת(יוסף, קובי), (קושיה שלישית).
לכן אב בשושלת(יעקב, קובי), (קושיה שניה).
לכן אב בשושלת(יצחק, קובי), (קושיה ראשונה).
לכן אב בשושלת(אברהם, קובי), **תשובה.**

איור 7.5 – קבלת התשובה על בסיס חוק רקורסיבי

כפי שרואים באיור, השאילתה מפנה אותנו אל החוק הראשון של אב_בשושלת, אשר מכוון אותנו אל החוק השני, אשר מזמין את החוק הראשון וכן הלאה. בדרך זו, אנו עוברים מאברהם אל יצחק, מיצחק אל יעקב, מיעקב אל הבא בתור, עד אשר מגיעים אל צחי, שהוא אביו של קובי. התהליך הרקורסיבי נעצר כאן על ידי החוק הראשון, שקובע כי מכיון שצחי הוא אביו של קובי, אזי הוא גם אב בשושלת שלו. מכאן מתחילה תנועה בכיוון חזרה: על פי החוק השני גם אביו של צחי, אבי, הוא אב בשושלת של קובי וכן הלאה. המסקנה: אברהם הוא אב בשושלת של קובי.

התכנית של פרולוג נקראת **רקורסיבית**, אם היא מכילה חוק רקורסיבי. נאמר לסיום, כי תכניות אשר נכתוב בהמשך הספר, תהינה ברובן תכניות רקורסיביות.

7.9 חוקים רקורסיביים – תרגול

בסעיף זה נציג לפניך שאלות, אשר בעזרתן תוכל להעריך את התקדמותך בלימוד חוק רקורסיבי. מומלץ לבצע את העבודה בשלבים הבאים:

1. לענות על השאלה מבלי להריץ את התכנית.
2. להריץ את התכנית ולהשוות את התשובות.
3. להיעזר ב-trace במידה והתשובה שעל הנייר שונה מזו שעל גבי המסך.

נדגיש, כי על כל אחת מהשאלות ניתן לענות מבלי להזדקק למחשב.

עבודה מהנה!

שאלה 1

ענה על השאילתות הבאות על בסיס התכנית "שושלת היוחסין של קובי" (תכנית 7.4).

להלן השאילתות:

א. ?- אב_בשושלת(X, קובי).
ב. ?- אב_בשושלת(אברהם, X).
ג. ?- אב_בשושלת(X, Y).

מצא את כל התשובות האפשריות לכל אחת מהשאילתות האלה.

שאלה 2

לפניך שתי שאלות:

1) מי הם אבות אבותיו של יוסף?
2) מי הם צאצאיו של יוסף?

א. תרגם את השאלות האלה לשאילתות של פרולוג.
ב. לכל אחת מהשאילתות, מצא את אוסף כל התשובות האפשריות על פי התכנית 7.4.
ג. תן הסבר לצעדי החישוב של השאילתה הראשונה בדומה לזה שבאיור 7.5.

להלן נציג כמה תכניות, אשר בהן נשתמש בחוק רקורסיבי. הראשונה מדגימה תהליך של הסקת מסקנות בשרשרת הבאה:

ים אם חם.
חם אם שמש.
שמש אם קיץ.

מפסוקי השרשרת אנו מסיקים כי ים אם קיץ. התכנית שלפנינו מתארת את תהליך הסקת המסקנה:

```
מסקנה_מיידית(ים, חם).
מסקנה_מיידית(חם, שמש).
מסקנה_מיידית(שמש, קיץ).
מסקנה(X, Y):- מסקנה_מיידית(X, Y).
מסקנה(X, Y):-מסקנה_מיידית(X, L),מסקנה(L, Y).
```

תכנית 7.5 – הסקת מסקנות

שאלה 3

מצא את כל התשובות לשאילתות אלו:

```
?- מסקנה(ים,קיץ).
?- מסקנה(X,קיץ).
?- מסקנה(ים,X).
?- מסקנה(X,Y).
```

הערה: ניתן להשתמש במונחים "תולדה מיידית" ו"תולדה" (אשר השתמשנו בהם בדיון בפרק 4), במקום המונחים "מסקנה מיידית" ו"מסקנה".

שאלה 4

א. עיין בתכנית לוגית "סיפור קיץ" שבמבוא לספר זה והצג את תהליך הסקת המסקנות בשרשרת באמצעות היחסים מסקנה_מיידית ו-מסקנה.

ב. מצא את כל התשובות לשאילתות אלו:

```
?- מסקנה(מבלים, קיץ).
?- מסקנה(עושים_חיים, קיץ).
?- מסקנה(מבלים, שוחים).
?- מסקנה(X, קיץ).
?- מסקנה(שוחים, X).
?- מסקנה(X, Y).
```

התכנית הבאה דנה במערכת כבישים בין-עירוניים. נביא את הגירסה האנגלית שלה. הישובים a ו-b נקראים **קשורים** אם יש כביש המקשר אותם:

```
connected(a,b):-road(a,b).
```

כמו-כן, הישובים a ו-b **קשורים**, אם קיימים הישובים שביניהם:

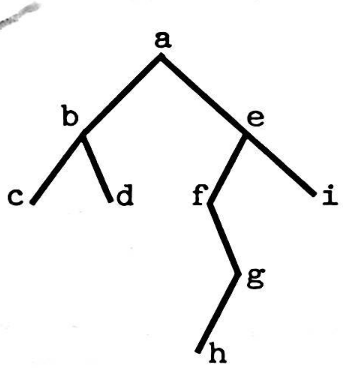
```
a, link1, link2, .... , b
```

הישוב a הוא הראשון בסידרה, הישוב b הוא האחרון בסידרה, ולכל זוג ישובים סמוכים (כמו a ו-link1, למשל, או כמו link1 ו-link2) יש כביש המקשר אותם. קשר מסוג זה נבטא באמצעות הגדרה רקורסיבית:

```
connected(a,b):-road(a,Link),connected(Link,b).
```

להלן התכנית בשלמותה.

```
road(a,b).
road(b,c).
road(b,d).
road(a,e).
road(e,f).
road(f,g).
road(g,h).
road(e,i).
connected(X,Y):-road(X,Y).
connected(X,Y):-road(X,Link),
                connected(Link,Y).
```



תכנית 7.6 – הקשר הבין-עירוני

הערה: בתכנית זו מניחים, כי הכבישים הם חד-סטריים. כלומר, הכביש שמקשר את a ו-b איננו מקשר את b ו-a.

שאלה 5

א. מצא את כל התשובות לשאילתה:

```
?- connected(e, X).
```

ב. מצא את כל התשובות לשאילתה:

```
?- connected(X, c).
```

ג. מצא את כל התשובות לשאילתה המורכבת:

```
?- connected(X, g), connected(X, i).
```

ד. מצא את כל התשובות לשאילתה המורכבת:

```
?- connected(a, X), connected(X, h).
```

תכנית 7.6 היא גירסה של הקשר הבין-עירוני, אשר תואר בסעיף ב' של ההקדמה. בסעיף 6 של חטיבה רביעית נציג תכנית נוספת בנושא זה.

שאלה 6

תאר את המחסנית LIFO בעזרת היחסים:

```
on(Item1, Item2).
above(Item1, Item2).
```

רמז: את היחס above/2 ניתן להגדיר באופן רקורסיבי על בסיס של היחס on/2.

שאלה 7

נבנה בסיס נתונים סביב היחס part_of/2:

```
part_of(Part, Object).
```

נוסיף לבסיס זה את החוקים:

```
is_in(Part, Object):-
                part_of(Part, Object).
is_in(Part, Object):-
                part_of(Part, Part1),
                is_in(Part1, Object).
```

בהתאם לבסיס הנתונים שלך בדוק מהן התשובות לשאילתה:

```
?- is_in(Part, Object).
```

תן הסבר לכל תשובה.

חטיבה שלישית –

תכנות לוגי בפרולוג

כתיבת תכנית בשפה פרוצדורלית היא הצפנת אלגוריתם, משמע, הצפנה של סדרת הוראות חד-משמעיות עם סדר ביצוע מוגדר. כתיבת תכנית לוגית היא תאור בעיה, ולכן היא נעשית ברמה אחרת: ברמה של יחסים, קשרים בין היחסים והסקות לוגיות. את היחסים מתארים באמצעות עובדות וחוקים, ומתוכם, באמצעות השאילתות, מסיקים את המסקנות. בהתאם לגישה זו מציגים את חישוב השאילתה כתהליך לוגי של הסקת מסקנות, מבלי לרדת לרמה של ביצוע, כלומר, מבלי לברר איך מערכת הפרולוג מרכיבה את התשובה. באופן מופשט ניתן לומר כך: תכנית לוגית מציינת **מה** לעשות, ומערכת הפרולוג מחליטה **איך** לעשות.
אנו לומדים אם-כן, כי בתכנות לוגי ניתן להפריד בין שתי רמות:

- **רמה לוגית**, המתבטאת בהגדרות של יחסים ושל קשרים ובניסוח של השאילתות.

- **רמת ביצוע**, אשר היא נחלתה של מערכת הפרולוג. כדי לענות על השאילתה, מערכת הפרולוג מאתרת עובדות וחוקים, מבצעת הצבות ובוחנת אלטרנטיבות, מבלי שבתכנית נכתב במפורש: "אתר", "הצב", "בחן".

על פי העקרון של תכנות לוגי, הרמה הלוגית היא אשר קובעת את הסמנטיקה, כלומר, את משמעות התכנית. האם הרמה הלוגית קובעת גם את הפלט?
במילים אחרות, האם ניתן לנבא את תוצאות השאילתה על-פי היחסים והקשרים הלוגיים, מבלי שנדע איך המערכת מאתרת, מציבה, בוחנת ומבצעת?

נעיין בשתי תכניות לוגיות, married1, married2, אותם הכרנו בסעיף 7.7. התכנית married1 מורכבת משני פסוקים. פסוקים אלה מרכיבים גם את התכנית married2, אלא שסדרם הפוך. למדנו כי סדר של כתיבת הפסוקים אינו משפיע על המשמעות של תכנית לוגית. ניתן לצפות שהתשובה לשאילתה ?-married1(X,Y) תהיה זהה לזו של השאילתה ?- married2(X,Y)? האמנם? ראינו כי לא כך הדבר.

לשאלה למה התשובות אינן זהות אין הסבר ברמה הלוגית, אך יש לה הסבר ברמה פרוצדורלית, כלומר, ברמה של ביצוע. נקבל אם-כן, כי ביישום עקרונות של תכנות לוגי באמצעות תכנות בפרולוג אי אפשר להתעלם מן הצד הפרוצדורלי, אותו יש להכיר אם ברצוננו לכתוב תכניות מעשיות.

היחס בין הצד הלוגי והצד הפרוצדורלי במערכת של פרולוג דומה ליחס בין הדרג המתכנן והדרג המבצע במערכת כלשהי: מבחינה לוגית התכנית יכולה להיות מושלמת, אך התוצאות תהיינה טובות או מאכזבות, בהתאם לביצוע. נקדים ונאמר כי אין לתלות את האשמה בשפת הפרולוג. מקור האכזבה טמון בכלי אשר באמצעותו אנו מיישמים את מערכת הפרולוג. כלי זה הוא המחשב של הדור הנוכחי. מחשב זה מבצע את חישוב השאילתה באופן סדרתי, פסוק אחר פסוק. לכן התכנית עלולה להיתקע בפסוק מסויים מבלי לדעת שהפסוק הבא "יציל את המצב". כנראה שהבעיה היתה נפתרת, לו ניתן היה לבחון את פסוקי התכנית במקביל על מנת לבחור את הפסוק המתאים. ביצוע מסוג זה שייך לטכנולוגיית הדור הבא. ומכאן, שאם רוצים לקצור היום את הפירות של טכנולוגיית העתיד ובכללם את שפת העתיד פרולוג, צריכים ללמוד את דרכי פעולתה של מערכת הפרולוג ברמת הביצוע וברמה הלוגית כאחד.

בחטיבה זו נלמד לתכנת בפרולוג. בפרק 8 נכיר את התחביר ואת מבנה הנתונים של פרולוג. לאלה מן הקוראים, אשר השלימו פרק 6, תהיה זו מלאכה קלה, שכן לוגיקת היחסים היא המקור של פרולוג. בפרק 9 נדון במבנה, הקרוי רשימה. הרשימה של פרולוג הינה מבנה רקורסיבי, ועל כן רוב התכניות של פרולוג אשר משתמשות ברשימות, מכילות חוקים רקורסיביים. בפרק 10 נציג את המודל החישובי של תכנית לוגית, ובפרקים 11 ו-12 נכיר את הפרדיקטים, המוגדרים בתוך המערכת והמכונים פרדיקטי המערכת או פרדיקטים המוגדרים מראש.

את היעד הלימודי – תכנות מעשי בפרולוג, נשלים באמצעים האלה: בפרק 8 נלמד לבנות בסיסי נתונים, ובפרק 9 נעסוק בכתיבה רקורסיבית. תחילה ננתח תכניות נתונות (סעיפים 9.3 ו-9.4), ולאחר מכן נעצב סדרה של תכניות מיון (סעיף 9.5). בפרק 10 נכיר את המודל החישובי של תכנית לוגית ואת יישומו על ידי מערכת הפרולוג, ובפרקים 11 ו-12 נלמד להיעזר בפרדיקטי המערכת לצורך ייעול התכנית.

פרק 8

כללי התחביר ומבנה הנתונים

בפרק זה נכיר את מבנה הנתונים של פרולוג (סעיפים: 8.1, 8.2, 8.3). נלמד לבנות בסיסי נתונים ולדלות מהם מידע באמצעות חוקים, אשר הוגדרו במיוחד למטרה זו (סעיף 8.4). נלמד לקרוא רשומות בפרולוג, ולהציגן באמצעות תרשים דמוי עץ (סעיפים: 8.3, 8.5). בנוסף, במסגרת פרק זה נכיר אופרטורים לחישוב והשוואה, נלמד את פעולתם ונעמוד על הבעיות הנוצרות עקב השימוש בהם (סעיף 8.6).

8.1 שפת פרולוג – סימנים וביטויים

האלף-בית של פרולוג מורכב מאותיות, ספרות וסמלים מיוחדים. בפרולוג האנגלי משתמשים באותיות גדולות וקטנות. בפרולוג העברי משתמשים באותיות עבריות וגם באותיות לועזיות גדולות. לפרולוג יש גירסאות שונות ובהתאם לכך יש גם הבדלים בסימון. בספר זה נשתמש בסמלים לפי גירסת WISDOM (ראה נספח א').

מסימני האלף-בית בונים את **ביטויי השפה** (terms). הביטויים הפשוטים ביותר של פרולוג בנויים **מאטומים** ו**מספרים**.

האטום של פרולוג שייך לאחת הקבוצות א, ב, ג, שלהלן:

א. אוסף אותיות וספרות שמתחיל באות קטנה ואינו כולל רווחים. את המילים באוסף זה כותבים ברצף (ללא רווחים), או מחברים באמצעות קו תחתון.

אטום	atom
הורה	parent
ההורה_של	the_parent_of
ב831	b34Ad

ב. אוסף סמלים כלשהו המוקף משני צדדיו בגרש. כשקיים חשש, כי סמל כלשהו אינו אטום, כדאי להקיף אותו בגרש משני צדדיו.

'האב של'	'the father'
'FILE'	'FILE'
'53A'	'53A'

ג. אוסף של סמלים מיוחדים. סמלים אלה שונים בגירסאות השונות:

```
=:=
:
=
>>>
```

בחלק מהגירסאות של פרולוג משתמשים במספרים טבעיים בלבד: 0, 1, 2, וכן הלאה. יש גירסאות בהן ניתן להשתמש בכל מספר שלם בין 32768-לבין 32768, ויש גירסאות בהן ניתן להשתמש בכל מספר ממשי.

האטומים והמספרים הם **קבועי השפה** (constants). בנוסף, ישנם **משתנים** (variables). המשתנים הם:

א. קו תחתון המכונה "משתנה אנונימי" (anonimous).

ב. אוסף סמלים המתחיל באות גדולה או בקו תחתון ללא רווחים.

_בן	Son
X	X
FATHER	_father
1D	DATA1
_כל_אחד	Anyone

הביטויים של השפה הם: קבועים, משתנים וביטויים מורכבים. השם האחר של ביטוי מורכב הוא **מבנה** (structure). המבנה של פרולוג בנוי **מפנקטור** (functor) **וארגומנטים** (arguments). הפנקטור הוא אטום. הארגומנטים הם אטומים, משתנים או מבנים. להלן דוגמאות:

1. אב(אברהם, יצחק)
פנקטור ארגומנט ארגומנט

father(abraham, isaac)
פנקטור ארגומנט ארגומנט

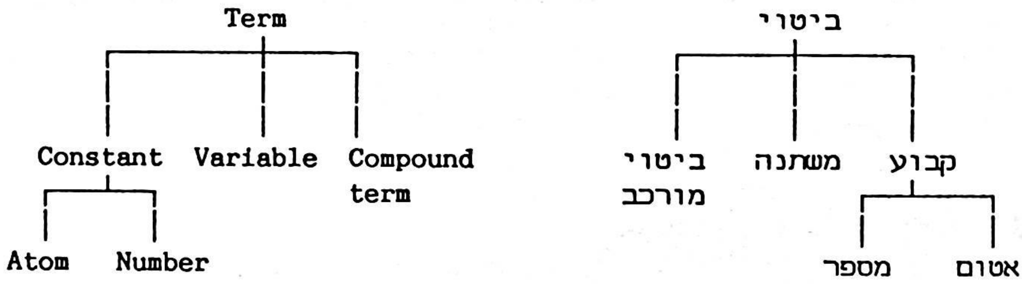
2. זוג(אב(X), אם(Y))
פנקטור ארגומנט ארגומנט

couple(father(X), mother(Y))
פנקטור ארגומנט ארגומנט

3. אוהב(דני, חברות(רותי, אתי))
פנקטור ארגומנט ארגומנט

likes(tom, sisters(ann, mary))
פנקטור ארגומנט ארגומנט

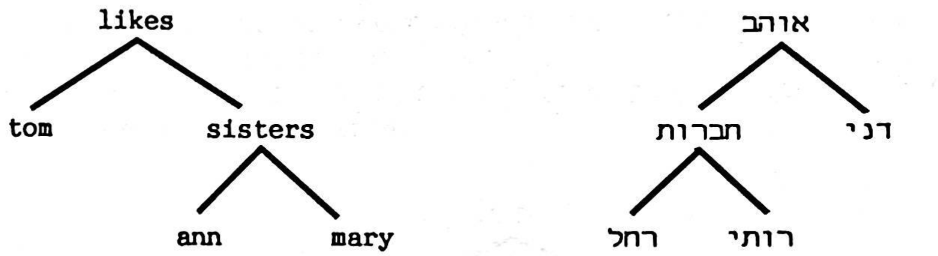
את הגדרת הביטוי נדגים בעזרת איור 8.1. הגדרה זו הינה רקורסיבית.



איור 8.1 – הגדרה של ביטוי

הביטוי מאופיין גם על ידי פנקטור וגם על ידי מספר הארגומנטים. משמע, הביטויים: אב(אברהם) ו-אב(אברהם, יצחק), הם ביטויים שונים לחלוטין. מספר הארגומנטים של הפנקטור מכונה בשם **רבב** או **אריטי** (arity) של פנקטור. הפנקטור אב בביטוי אב(אברהם) הוא של אריטי 1 והוא מסומן כך: אב / 1. הפנקטור אב של הביטוי השני הוא של אריטי 2 ומסמנים אותו כך: אב / 2. בפרולוג האנגלי הסימונים של (father(abraham ושל (father(abraham, isaac הם father/1 ו-father/2 בהתאמה.

בביטוי כדוגמת ((likes(tom, sisters(ann, mary הפנקטור tom נקרא **הפנקטור הראשי** (principal functor). מטעמים טכניים מכנים את הקבוע בשם **פנקטור של אריטי 0**. למשל, את הקבוע חם נסמן כך: חם / 0, ואת הקבוע atom נסמן כך: atom/0.
ביטוי מורכב של פרולוג ניתן לתאר באמצעות תרשים הקרוי עץ, כדוגמת זה שבאיור 8.2. לנושא הצגת הביטוי באמצעות עץ יוקדש הסעיף הבא.



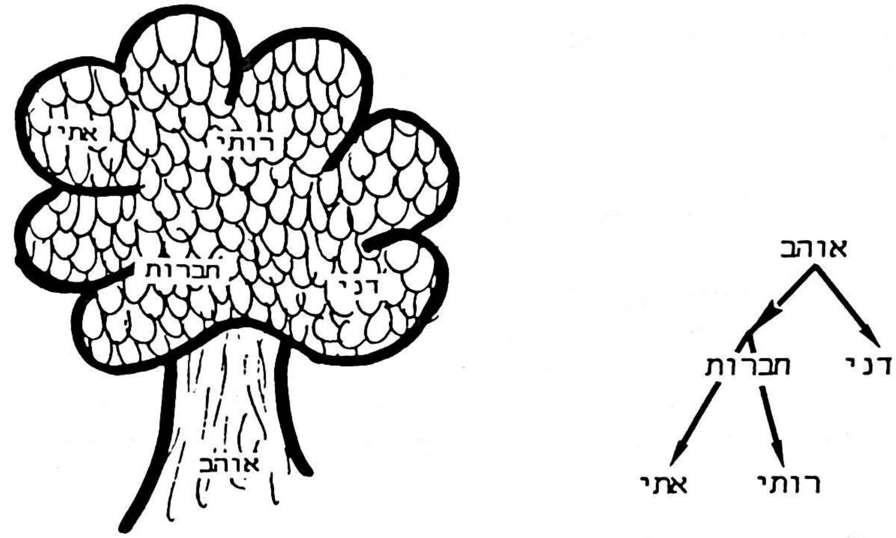
likes(tom, sisters(ann, mary))   אוהב(דני, חברות(רותי, רחל))

איור 8.2 – תאור הביטוי באמצעות עץ

8.2 עצים

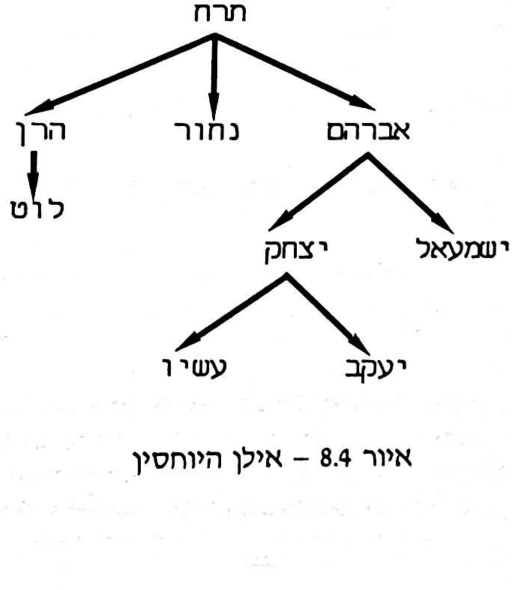
ביטויים מורכבים של פרולוג ניתן לתאר בצורה מדורגת. ניקח למשל, את הביטוי הבא: אוהב(דני, חברות(רותי, אתי)). את התאור המדורג של הביטוי הזה נתחיל מהפנקטור הראשי אוהב. אחריו באים הארגומנטים לפי סדר הופעתם: דני ו-חברות(רותי, אתי). לארגומנט השני יש **מדרג** (hierarchy) משלו: הפנקטור חברות והארגומנטים רותי, אתי.

כדי "לראות" כנגד עינינו את מדרג הביטוי, אנחנו נעזרים בתרשים דמוי עץ. לנוחותנו נהפוך את העץ ונתחיל לציירו מן השורש (ראה איור 8.3). בשורש העץ נרשום את הפנקטור הראשי אוהב. בעקבותיו יופיעו שני הארגומנטים, כאשר השני שבהם נרשם בהתאם למדרג הפנימי שלו.



איור 8.3 – תרשים דמוי עץ של מבנה מורכב

במהלך הספר כבר נעזרנו בתרשים מסוג זה. השתמשנו בו כשהגדרנו את הביטוי של לוגיקת היחסים (ראה איור 6.1). גם תיארנו בעזרתו את החוק, המגדיר יחס סב (ראה איור 7.1). המשותף לכל המקרים שבהם נעזרנו בתרשים דמוי עץ, היה המבנה המדורג של העצם המתואר. קשה היה לתאר את העצם בעזרת מערך סדרתי. מאידך, "ראינו" את העצם על כל חלקיו, כאשר נעזרנו במערך כללי יותר, משמע בתרשים דמוי עץ.
את היעילות של ייצוג דמוי עץ ניתן לראות בתאור קשרי אב-בן, המוכר בכינוי "אילן יוחסין". להלן אילן היוחסין של המשפחה התנכי"ת:



איור 8.4 – אילן היוחסין

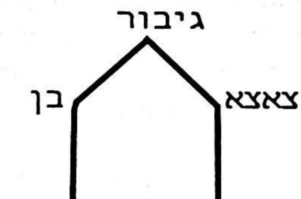
מעתה נשתמש בייצוג דמוי עץ ובשל כך נקדיש לנושא זה את הדיון שלהלן. תחילה נבחן את המאפיינים של התרשימים הקרויים **עצים**.

העץ מורכב מ**צמתים** (nodes). לעץ שבאיור 8.3, יש חמישה צמתים: אוהב, דני, חברות, רותי, אתי. לעץ שבאיור 8.4 יש תשעה צמתים, המייצגים את השמות של בני המשפחה התנכי"ת. זוגות של צמתים מחוברים יחד בעזרת **קשתות** (arcs). למשל באיור 8.4, הצומת בשם יצחק מחובר על ידי הקשתות עם הצמתים האלה: אברהם, יעקב, עשיו. כל שני צמתים של העץ קשורים באמצעות **נתיב** (path), המורכב מקשתות. למשל, מצומת בשם נחור ניתן להגיע אל הצומת יעקב בנתיב העובר דרך הצמתים תרח, אברהם, יצחק.

כל זוג צמתים, שמחובר בעזרת קשת, נמצא בקשר משפחתי. האחד מבני הזוג מכונה בשם **הורה** (parent). השני מכונה **עוקב** (successor), או **בן**. ניקח לדוגמה, את הזוג אברהם ו-יצחק. מן ההקשר התנכ"י אנו יודעים, כי הצומת אברהם הוא ההורה והצומת יצחק הוא העוקב, או הבן. ניקח זוג אחר של צמתים: יצחק ו-יעקב. הצומת יצחק הוא ההורה והצומת יעקב הוא העוקב. אבל אל נא נתעלם לרגע מן ההקשר התנכ"י. איך נדע מי מבני הזוג הוא ההורה ומי העוקב? כדי להבדיל בין ההורה ובין העוקב נגדיר את מדרג הצמתים. בכל עץ יש צומת אחד ויחיד המכונה בשם **שורש** (root) והמאופיין על-ידי התכונה הבאה: אין לו הורים. בלשון אחרת, לעץ יש צומת אחד ויחיד, שמופיע בכל אחד מקשריו עם צמתים אחרים בתור הורה ושאינו מופיע בתור עוקב. את צומת העץ שהוכרז כשורש, מציירים בדרך כלל בראש העץ, את בניו מסדרים **במפלס** (level) מתחת לשורש, את בניהם של בניו מארגנים במפלס הבא, וכן הלאה.

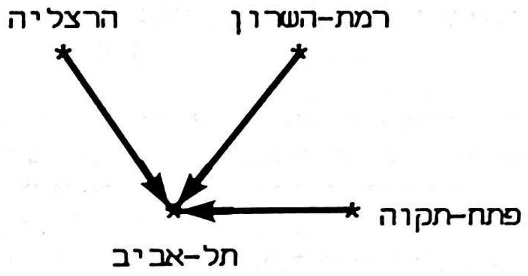
עבור צומת מסוים, הבנים, בני-בנים, בני-בני-בנים וכן הלאה, מכונים **צאצאים** (descendants) של הצומת. הצומת עצמו מכונה **האב הקדמון** (ancestor) של כל אחד מצאצאיו. אשר על כן, העץ הינו מבנה **מכוון**. הכיוון נקבע על-ידי הקשר אב-בן. באיורים, כיוון זה מסומן לעיתים באמצעות **חץ** בקצה הקשת (ראה איור 8.4).

למבנה הקרוי עץ יש תכונה נוספת: לכל עוקב יש הורה אחד בלבד. פירוש הדבר הוא שבעץ אין **לולאות** (loops). באחד מספרי המדע הבדיוני מופיע הסיפור הבא: גיבור הסיפור מגיע במכונת זמן מתוחכמת משנת 2120 אל שנת 1970. הוא פוגש את אשת חלומותיו, אשר מולידה לו בן. הגיבור חוזר אל שנת 2120 ואז מתברר לו, כי הוא אחד מצאצאיו של הבן שלו. אם נרצה לצייר את שושלת היוחסין של הגיבור הזה, נקבל את הלולאה שבאיור 8.5.



איור 8.5 – לולאת היוחסין של הגיבור הבדיוני

שרשרת האבות של גיבור הסיפור תוארה באמצעות מבנה המכונה **גרף** (graph), אשר יש לו צמתים וקשתות, אבל אין לו בהכרח שורש. גם המגבלה שלכל עוקב יהיה הורה אחד בלבד אינה קיימת בגרפים. משמע, שלצומת מסוים של הגרף ישנם לפעמים שני אבות ויותר. בגרף, אשר באיור 8.6, יש לצומת תל-אביב שלושה אבות: הרצליה, רמת-השרון, פתח-תקוה.



איור 8.6 – גרף שמתאר קשרים בין-עירוניים, המכוונים על-פי העקרון אב-בן.

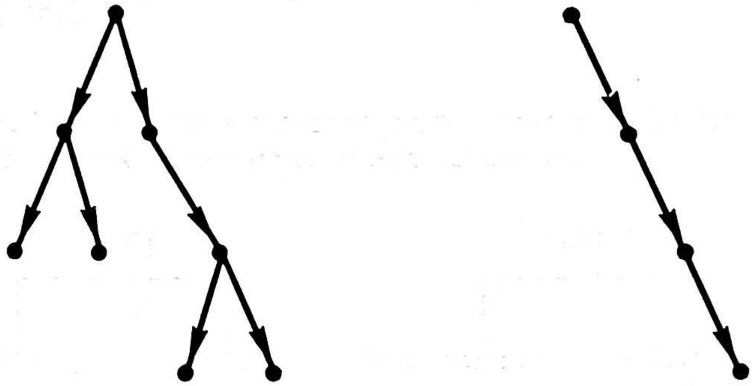
נציין תכונה נוספת של עצים: בעץ ישנם צמתים, אשר אין להם בנים. צמתים אלו מכונים **עלים** (leaves). בעץ שבאיור 8.4 הצומת לוט הוא עלה, וכמו כן גם הצומת נחור. גם בגרף ישנם עלים: הצומת תל-אביב של הגרף שבאיור 8.6 הינו עלה.

את הנאמר לעיל נסכם כך: עץ הוא אוסף של צמתים המחוברים בעזרת קשתות. לשני צמתים המחוברים על ידי הקשת ישנם תפקידים: האחד הוא ההורה, השני הוא העוקב, או הבן. לכל עוקב יש הורה אחד בלבד. קיים צומת אחד ויחיד שאיננו מופיע בתפקיד של עוקב: צומת השורש. צומת שאין לו בנים, נקרא **עלה** (leaf). חלק של העץ, המורכב מהורה על כל צאצאיו, מכונה בשם **ענף** (subtree), או **שבט**. את הצמתים שבעץ ניתן לארגן ב**שרשרת** (chain) על פי הקשר המשפחתי הורה-בן-נכד-, וכן הלאה.

העץ נקרא **אינסופי** (infinite) אם יש בו שרשרת אשר אינה מסתיימת. אומרים שהשרשרת היא **בעלת אורך** N, אם בנוסף להורה יש בה N צאצאים. כל זוג צמתים ניתן לקשור באמצעות נתיב, אבל לא כל זוג ניתן לקשור באמצעות שרשרת משפחתית. לדוגמה, בעץ שבאיור 8.4, יש נתיב המחבר את צומת לוט עם צומת ישמעאל, אבל אין שרשרת שתחבר אותם, משום שאף לא אחד מהם הוא האב הקדמון של האחר. מאידך השורש הוא האב-הקדמון של כל צומת אחר ולכן, כל צומת ניתן לחבר עם השורש באמצעות שרשרת משפחתית. עבור כל צומת נתון, קיימת שרשרת אחת בלבד, המקשרת בינו לבין השורש. דבר זה נובע מן העובדה שלכל בן יש הורה אחד בלבד.

אומרים שהצומת הנתון **נמצא בעומק** N, אם אורך השרשרת שמקשרת בין השורש לבין הצומת הזה הוא N. בניו של אותו הורה מכונים **אחים**: המאפיין של האחים הוא, שכולם נמצאים באותו עומק. אומרים כי **עומק העץ** (depth of the tree) הינו N, אם השרשרת הארוכה ביותר, שמקשרת בין השורש לבין אחד מעלי העץ היא באורך N. לדוגמה, עומק העץ שבאיור 8.3 הינו 2, ועומק העץ שבאיור 8.4 הינו 3.

תיאורטית, ניתן להגדיר עץ, שאין בו צמתים בכלל. עץ זה מכונה בשם **עץ ריק** (empty tree). קיים גם עץ שעומקו 0, משמע הוא בעל צומת אחד בלבד. מערך סדרתי גם הוא עץ: האיבר הראשון בו הוא שורש, השני הוא בנו וכן הלאה. האיבר האחרון של הסדרה הוא עלה (ראה איור 8.7). עץ אשר בו להורה יש לכל היותר שני בנים מכונה **עץ בינרי** (binary tree). בסעיף הבא נציג תכנית המתארת עץ בינרי. ישנם גם סוגי עץ אחרים, אשר לא נדון בהם.

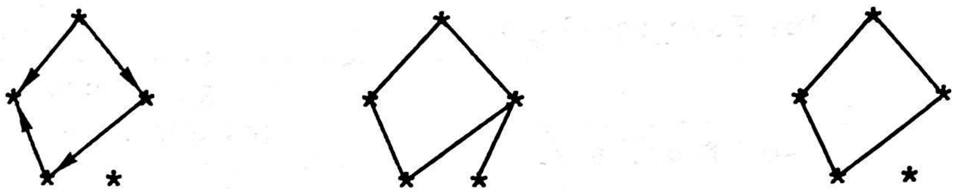


(א) סדרה (ב) עץ בינרי

איור 8.7 – סוגים שונים של עצים: (א) סדרה, (ב) עץ בינרי

נאמר לסיום, שעץ הוא מבנה פורמלי אשר משתמשים בו כאשר יש צורך לתאר מספר אירועים "במקביל". נשתמש בעץ, אם נרצה לתאר "במקביל" את כל בניו של תרח, או את כל הארגומנטים של הפנקטור הנתון. קיים מבנה פורמלי אחר לתיאור אירועים "במקביל" – **גרף**. כל עץ הוא גם גרף, אך ההיפך אינו נכון. נציין גם, כי בספרי לימוד שונים יכולים להיות הבדלים במינוח ובניסוח של ההגדרות. בנוגע לגרפים, נשתמש במינוח הבא:

* גרף – אוסף של צמתים, אשר חלקם קשורים.
* גרף קשיר (connected)- גרף אשר כל אחד מצמתיו קשור לצומת אחר.
* גרף מכוון (directed, digraph)- גרף אשר בו כל זוג של צמתים קשורים מוגדר באמצעות הקשר אב-בן.

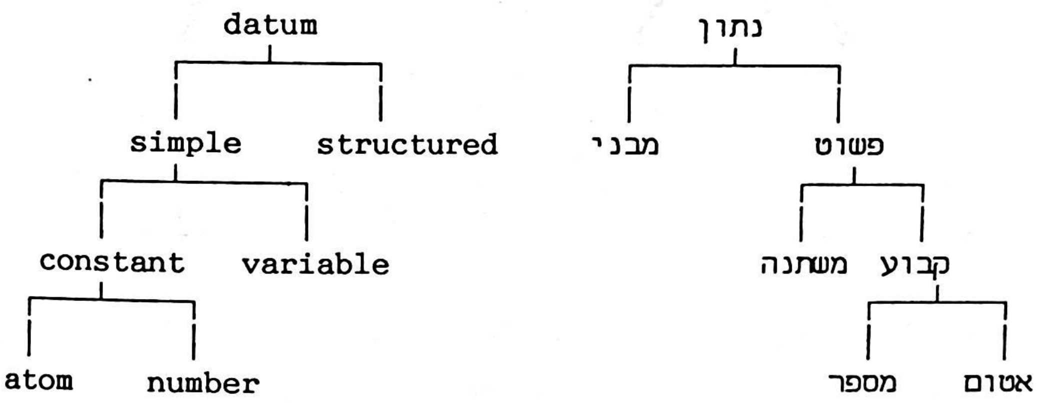


(א) גרף לא קשיר ולא מכוון (ב) גרף קשיר ולא מכוון (ג) גרף לא קשיר ומכוון

איור 8.8 – סוגים שונים של גרפים:
(א) לא קשיר, לא מכוון,
(ב) קשיר, לא מכוון,
(ג) לא קשיר, מכוון.

8.3 מבנים מורכבים

מביטויי השפה בונים את העובדות והחוקים ובהתאם לסוג הביטוי מציגים מבנה נתונים פשוט או מורכב. התרשים שבאיור 8.9 מתאר את מבנה הנתונים בפרולוג:



איור 8.9 – מבני הנתונים של פרולוג

השימוש במבני נתונים מורכבים מאפשר גמישות רבה בכתיבת תכניות. ניתן אם-כן, לכתוב תכנית בצורה מודולרית כך, ששינוי קל בנתון זה או אחר לא יחייב שינוי של התכנית. השימוש במבני נתונים מורכבים מאפשר גם גמישות בהצגת השאילתות. נעיין בדוגמאות אחדות.

דוגמה 8.1

את העובדה כי 2+4=6 ניתן לבטא בעזרת ביטוי פשוט:

```
two_plus_four_equals_six.
```

או בעזרת ביטוי מורכב:

```
plus(two, four, six).
```

או בעזרת ביטוי מורכב, בו צומצם מספר הארגומנטים לשניים:

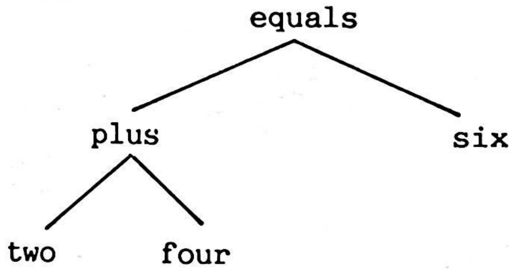
```
equals(plus(two, four), six).
```

בביטוי האחרון אסור להחליף את הפרדיקט plus במשתנה, אבל מותר להחליף במשתנה את הארגומנט plus(two,four) ואת כל אחד מהארגומנטים הבאים: two, four, six. בלשון אחרת: אם נציג את הביטוי בעזרת עץ, ניתן יהיה להחליף במשתנה את הענף plus(two,four) ואת כל אחד מהעלים two, four, six. אסור להציב משתנה במקום צומת פנימי, או במקום שורש. להלן השאילתות והתשובות:

```
?- equals(plus(X, Y), Z).
X = two
Y = four
Z = six
```

```
?- equals(X, Y).
X=plus(two, four)
Y=six
```

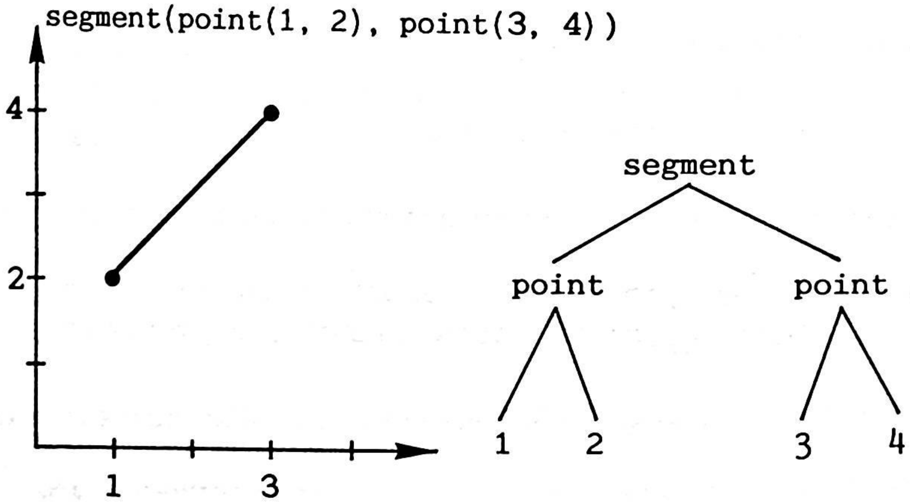
נציין, כי בתשובה לשאילתה ?-cquals(X,_) נקבל את ענף העץ, אשר שורשו plus:
X = plus(two, four).



איור 8.10 – עץ של equals/2

דוגמה 8.2

ניתן לבטא קטע של מישור באמצעות המבנה הבא:



(ב) קטע במישור (א) עץ של segment/2

איור 8.11 – קטע: (א) כביטוי מורכב של פרולוג,
(ב) כחלק של המישור.

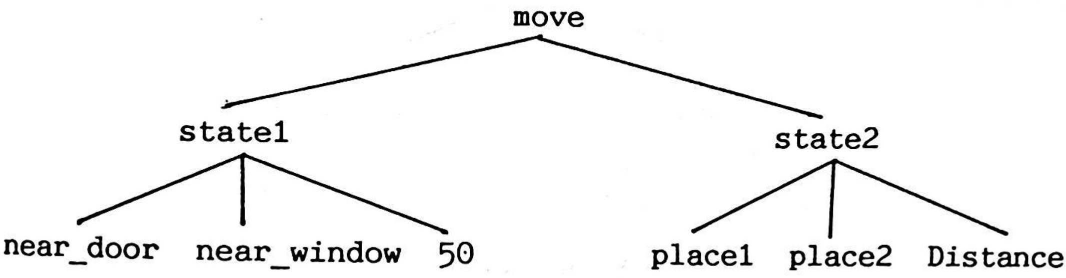
נציין, כי בתשובה לשאילתה: ?-segment(X,_) נקבל ענף שלם של העץ: X=point(1,2). בתשובה לשאילתה: ?-segment(point(X,_),_) נקבל את העלה: X = 1.

דוגמה 8.3

ניתן לבטא את התנועות של שני רובוטים ואת המרחק ביניהם בדרך זו:

```
move(state1(near_door, near_window, 50),
            state2(place1, place2, Distance)).
```

במצב state1 נמצא הרובוט הראשון במקום המכונה near_door והרובוט השני – במקום המכונה near_window. המרחק בין הרובוטים שווה ל-50 יחידות אורך כלשהן. במצב state2 נמצא הרובוט הראשון במקום place1 והשני – במקום place2, המרחק בין הרובוטים מסומן באמצעות המשתנה Distance.



איור 8.12 – עץ של move/2

גם כאן, בתשובה לשאילתה ?-move(X,_) נקבל את הענף:

```
X=state1(near_door, near_window, 50)
```

דוגמה 8.4

להלן חלק של רשומת (record) המשפחות: שם האב, שם האם ומספר הילדים:

```
משפחה(אב(ישראלי, ישראל), אם(ישראלי, ישראלה), ילדים(3)).
משפחה(אב(צבר, יורם), אם(צבר, רינה), ילדים(0)).
```

האיור 8.13 מתאר באמצעות עץ את הביטוי השני שברשומה.

נציג את השאילתה הבאה:

```
?- משפחה(_אב, _, _).
```

נקבל את התשובה:

```
_אב = אב(ישראלי, ישראל);
_אב = אב(צבר, יורם).
```

נציג שאילתה נוספת:

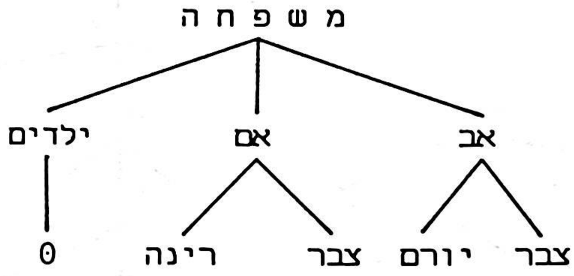
```
?- משפחה(אב(_משפחה, _פרטי), _, _).
```

נקבל:

```
_משפחה = ישראלי
_פרטי = ישראל
```

וכן הלאה.

הערה: נזכירכם, כי _אב הוא סימון נוסף של משתנה בפרולוג העברי.



איור 8.13 – עץ של משפחה / 3

דוגמה 8.5

הביטוי הבא הוא חלק מבסיס נתונים המתאר מערכת שעות לימודים במוסד כלשהו. נציג אותו תחילה בעזרת עץ, ואח"כ נכתוב אותו בתעתיק של פרולוג באמצעות שיטה המכונה **סריקת עץ**. הפרדיקט הראשי של הביטוי הוא קורס. המאפיינים של הקורס הם כינויו, זמן ומקום העברתו, ושמותיהם של המרצה והמתרגל. הזמן, המקום ושמות המרצה והמתרגל משתנים בהתאם לסמסטר שבו מלמדים את הקורס הזה, אף-על-פי שאת המערכת בונים בדרך כלל לשנה. לכן, למען המודולריות, הוחלט על מבנה הנתונים הבא:

קורס(_שם, _סמסטר1, _סמסטר2).

את הפרטים הנוספים הוחלט לציין בענף שבראשו נמצא הסמסטר המתאים:

סמסטר(_סמסטר, _הרצאה, _תרגול).

רוב הקורסים הועברו פעמיים בשבוע ולכן מבנה של הנתון הרצאה היה כדלהלן:

הרצאה(_מרצה, _יום1, _יום2).

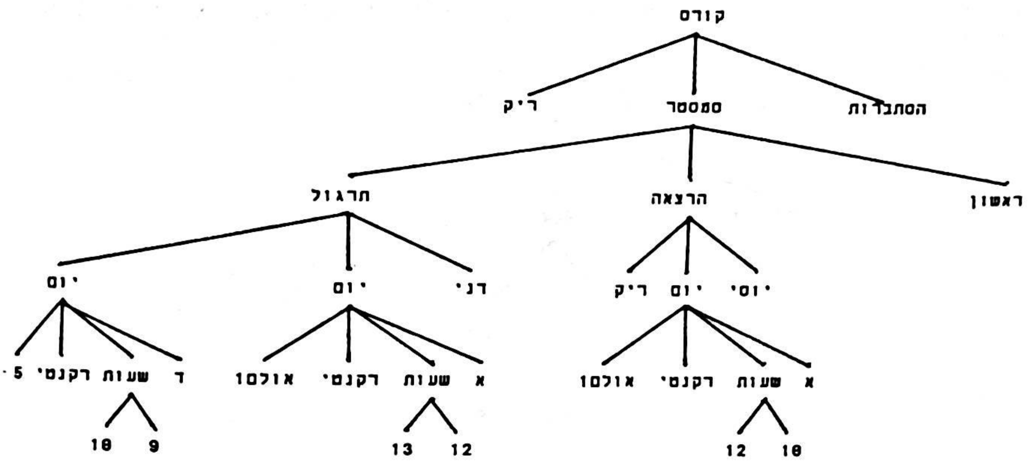
הנתון יום היה מורכב מיום של השבוע, שעות התחלה וסיום של ההרצאה, שם הבנין ומספר החדר:

יום(_יום, _שעות, _בנין, _חדר).

השעות תוארו באמצעות הביטוי:

שעות(_התחלה, _סיום).

לגבי ההרצאות שהתקיימו פעם בשבוע בלבד, סוכם היה שאת הפעם השניה, החסרה, נציין באמצעות אטום ריק.
העץ שמתאר את מערכת השעות של הקורס להסתברות הוצג באיור 8.14



איור 8.14 - עץ של קורס / 3

העץ מתאר את הנתונים האלה: קורס להסתברות מועבר בסמסטר הראשון ואינו מועבר בסמסטר השני. ההרצאה מתקיימת בימי א' של השבוע, בין השעות 10 עד 12, באולם 1 של בנין רקנטי. המרצה הוא יוסי. התרגילים מועברים על ידי דני בימים א' ו-ד', בבנין רקנטי. ביום א' התרגיל מתקיים מיד אחרי ההרצאה, וביום ד' הוא מתקיים מ-9 עד 10 בחדר מספר 5.

נכתוב את נתוני קורס ההסתברות כביטוי מורכב של פרולוג.

קורס(הסתברות, סמסטר(ראשון, הרצאה(יוסי, יום(א, שעות(10, 12), רקנטי, אולם1), ריק), תרגול(דני, יום(א, שעות(12, 13), רקנטי, אולם1), יום(ד, שעות(9, 10), רקנטי, 5))), ריק).

לצורך כתיבת הביטוי אנו סורקים את העץ מימין לעומק, ועוברים את הצמתים הבאים: קורס, הסתברות, סמסטר, ראשון, הרצאה, יוסי, וכן הלאה. כשיורדים למפלס נמוך יותר, פותחים את הסוגריים, וכשחוזרים, סוגרים אותם. לדוגמה:

יום(א, שעות(10, 12), רקנטי, אולם1) וכן הלאה.

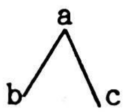
שיטה זו מאפשרת כתיבת קוד יעילה ומהירה. את בדיקת הסוגריים ניתן לעשות באמצעות האלגוריתמים הרגילים בתוך ענף כלשהו, או בתוך העץ כולו.

נעיר כי את סריקת העץ בפרולוג האנגלי יש לבצע משמאל לימין (ראה את שאלה 2 בסעיף הבא).

דוגמה 8.6

את המושג עץ בינרי (ראה סעיף 8.2) מגדירים רקורסיבית כך: עץ בינרי הוא עץ ריק, או עץ אשר כל אחד מענפיו הינו עץ בינרי. הביטוי nil משמש לסימון של עץ ריק. ניתן לתאר עץ בינרי באמצעות היחס tree(Left, Root, Right), שבו Root הינו השורש, ו-Left, Right הינם הענף השמאלי והימני בהתאמה, אשר שורשיהם הם בנים של השורש.
ניתן לתאר את העץ הבינרי אשר להלן, באמצעות היחס tree בדרך זו:

```
tree(tree(nil, b, nil), a, tree(nil, c, nil)).
```



התכנית הבאה בודקת האם הצומת X שייך לעץ הבינרי tree:

```
in(X, tree(_, X, _)).
in(X, tree(L, _, _)) :- in(X, L).
in(X, tree(_, _, R)) :- in(X, R).
```

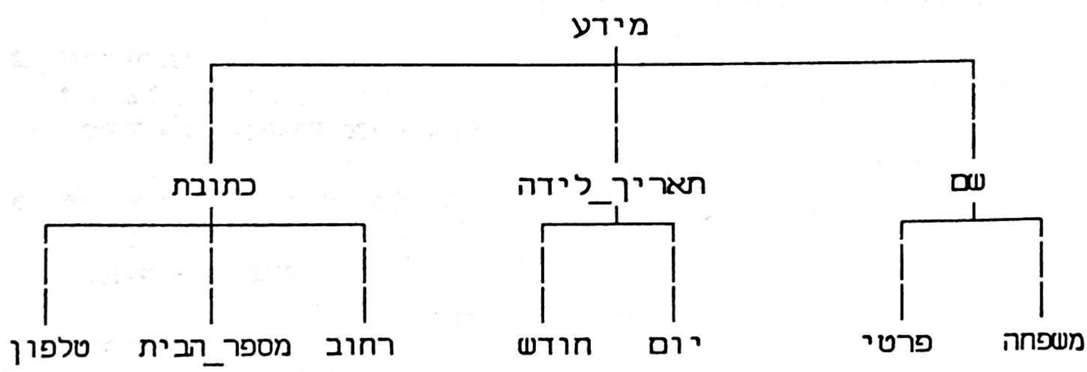
תכנית 8.1 – חיפוש צומת בעץ בינרי

התכנית מורכבת משלושה חוקים. החוק הראשון קובע כי הצומת X שייך לעץ אם הוא השורש. החוקים השני והשלישי הינם רקורסיביים: הצומת X שייך לעץ בינרי, אם הוא שייך לענף השמאלי או הימני שלו.

8.4 בניית קובץ נתונים – אירוע

בדוגמת בסיס הנתונים שנבנה על ידי חברי חוג פרולוג בבית הספר התיכון על-שם גינצבורג ביבנה, נראה את אשר ניתן לעשות באמצעות מבנה מורכב. נציין, כי להצגת המבנה בצורת עץ היתה חשיבות רבה במהלך העבודה.

המטלה היתה לבנות את קובץ נתוני הכיתה, אשר ממנו ניתן יהיה לדלות מידע על ימי הולדת, כתובות, מספרי זהות, מספרי טלפון וכן הלאה. הוחלט לבחור את הפרדיקט מידע בתור הפנקטור הראשי. בהמשך, נציג גירסה מקוצרת של נתוני הקובץ, הכוללת את שם התלמיד, תאריך-לידה וכתובת. השם מורכב משם משפחה ושם פרטי, תאריך-הלידה כולל יום וחודש, והכתובת כוללת את שם הרחוב, מספר הבית ומספר הטלפון. העץ שבאיור 8.15 מתאר את הביטוי המורכב של הקובץ "מידע".



איור 8.15 – עץ של מידע / 3

הביטוי המורכב, אשר כולל את המידע, נראה כך:

```
מידע(שם(_משפחה, _פרטי), תאריך_לידה(_יום, _חודש),
כתובת(_רחוב, _מספר_הבית, _טלפון)).
```

כדי להקל על דליית המידע, נבנו מספר חוקים. "חוק ימי הולדת" כלל את המידע הרלוונטי באמצעות הארגומנטים שם / 2 ו-תאריך_לידה / 2. את הביטוי כתובת / 3 החליף המשתנה האנונימי:

```
יום הולדת(_משפחה, _פרטי, _יום, _חודש):-
מידע(שם(_משפחה, _פרטי), תאריך_לידה(_יום, _חודש), _).
```

נציין את המודלוריות של "חוק ימי הולדת". התלמיד יכול לשנות את הכתובת. המידע הרלוונטי יעודכן מבלי לגרור את עדכון "חוק ימי הולדת".

התלמידים החליטו לערוך אחת לחודש מסיבה לכל החברים, אשר נולדו באותו חודש. כדי לקבל את שמות "בני ההולדת" נבנה החוק הבא:

```
מסיבת_הכיתה(_פרטי, _חודש):-
מידע(שם(_, _פרטי), תאריך_לידה(_, _חודש), _).
```

להלן חלק של התכנית "מידע". השמות הם שמות אמיתיים של חברי חוג פרולוג ביבנה, אך שאר הפרטים שונו מטעמים מובנים.

החוקים של תכנית 8.2 נועדו להקל על המשתמש להציג את השאילתות. נוכל לראות, כי הצגת שאילתה עם הפנקטור מידע / 3 מחייבת כתיבה רבה בשל המבנה המורכב שלו. דליית הנתונים הינה פשוטה, קלה ואלגנטית כאשר משתמשים בחוקי פרולוג שנכתבו.

הינה מספר דוגמאות של שאילתות:

1. מי נולד באפריל?

?-מסיבת_הכיתה(_פרטי, אפריל).
_פרטי = רמי;
_פרטי = חדוה.

2. איפה גר איל?

?-מען(איל, _כתובת).
_כתובת = כתובת(האורן, 130, 541423).

3. מה הוא שם המשפחה של יובל?

?-שם_מלא(יובל, _משפחה).
_משפחה = אבנשטיין.

4. מי גר ליד סיגל?

?-שכנים(סיגל, _פרטי, צאלון).
_פרטי= נמרוד.

```
מידע(שם(אהרון, סיגל), תאריך_לידה(21, נובמבר),
                כתובת(צאלון, 150, 551143)).
מידע(שם(ברטוב, רמי), תאריך_לידה(25, אפריל),
                כתובת(הדרור, 99, 531125)).
מידע(שם(מנור, איל), תאריך_לידה(2, ספטמבר),
                כתובת(האורן, 130, 54423)).
מידע(שם(ביירון, אביאור), תאריך_לידה(2, מרס),
                כתובת(הגפן, 37, 521114 )).
מידע(שם(דרור, צהלה), תאריך_לידה(5, אוגוסט),
                כתובת(האורן, 97, 521003)).
מידע(שם(וקס, נמרוד), תאריך_לידה(14, יוני),
                כתובת(צאלון, 100, 510341)).
מידע(שם(חרמון, עופר), תאריך_לידה(10, אפריל),
                כתובת(הגפן, 21, 550012)).
מידע(שם(מדר, חדוה), תאריך_לידה(17, אפריל),
                כתובת(ברניצקי, 99, 560103)).
מידע(שם(אבנשטיין, יובל), תאריך_לידה(15, מרס),
                כתובת(הדרור, 77, 540132)).


יום_הולדת(_פרטי, _יום, _חודש):-
   מידע(שם(_, _פרטי), תאריך_לידה(_יום, _חודש), _).

מספר_הטלפון(_פרטי, _טלפון):-
   מידע(שם(_, _פרטי), _,כתובת(_, _, _טלפון)).

מסיבת_הכיתה(_פרטי, _חודש):-
   מידע(שם(_, _פרטי),תאריך_לידה(_, _חודש), _)).

מען(_פרטי, _כתובת):-
   מידע(שם(_, _פרטי), _, _כתובת).

שכנים(_פרטי1, _פרטי2, _רחוב):-
   מידע(שם(_, _פרטי1), _, כתובת(_רחוב, _, _)).
   מידע(שם(_, _פרטי2), _, כתובת(_רחוב, _, _)).

שם_מלא(_פרטי, _משפחה):-
   מידע(שם(_משפחה, _פרטי), _, _).
```

תכנית 8.2 – מידע

8.5 בנייה של ביטויים מורכבים – תרגול

סעיף זה נועד לתרגול עצמי. מטרתו הינה להקנות מיומנות בעבודה עם נתונים מבניים ובסיסי נתונים.

אין צורך במחשב. הכן עט וניר.
עבודה מהנה!

שאלה 1

לפניך ביטוי מורכב המתאר את המתכנת John Smith, אשר עובד במקצוע החל משנת 1980 ומכיר את שפות התכנות אסמבלר 86 (ניסיון של שבע שנים), ופרולוג (ניסיון של שנתיים).

```
record(person(smith, john),
       works_from(12, january, 1980),
       prof_status(programmer(assembler86(7), prolog(2))).
```

א. תאר את הביטוי המורכב הזה באמצעות עץ.
ב. מצא את האריטי של הפנקטורים האלה:

1) record
2) works_from
3) prof_status
4) programmer

ג. ענה על השאילתות האלו:

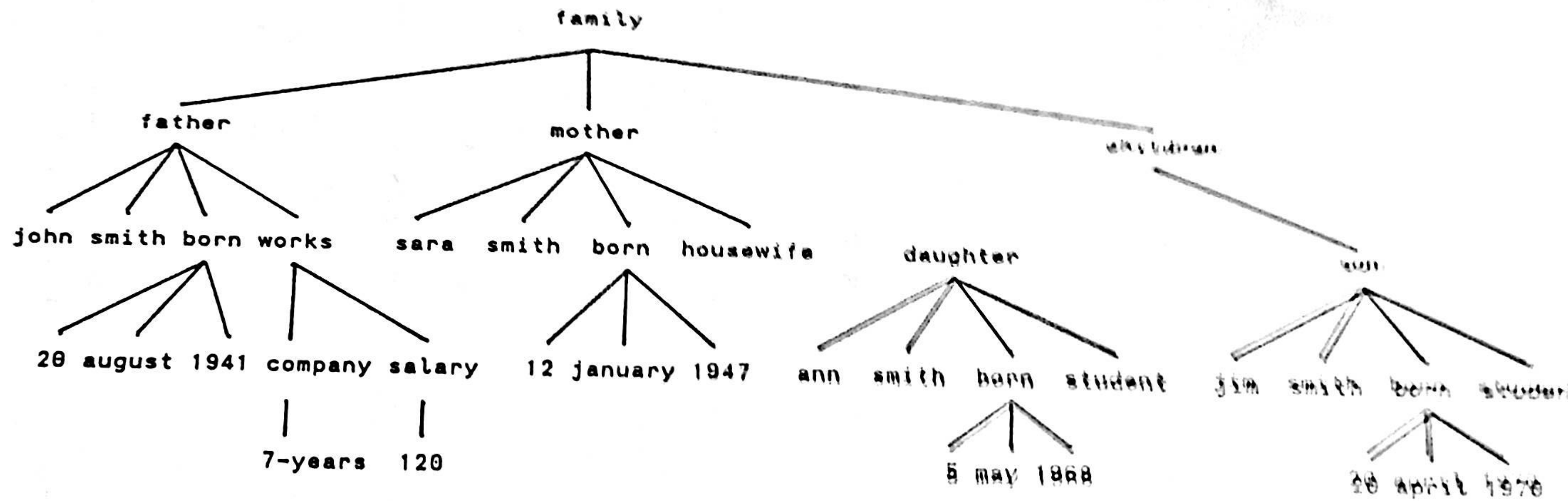
```
?- record(Person, Date, Profession).
? -record(_, work_from(_, _, year), Profession).
?- record(Person, _, prof_status(programmer(X, Y))).
?- record(person(Family, Name), _, prof_status(X)).
```

ד. נניח כי העובדה person/3 היא חלק מבסיס נתונים, בו נמצאים שמות נוספים של עובדים. המבנה של כל אחת מהעובדות בבסיס הנתונים זהה למבנה של העובדה person/3 דלעיל. אנו רוצים לדלות מידע מתוך בסיס הנתונים הזה. הצג בפרולוג את השאלות האלו:

1. הכן שמות של עובדים עם תאריכי תחילת עבודתם.
2. הכן שמות של מתכנתים.
3. הכן את שמות העובדים ואת מקצועותיהם.
4. הכן את שמות העובדים היודעים לתכנת בפרולוג.

שאלה 2

הנתונים של משפחת Smith מוצגים באיור 8.16 בצורה של עץ.



איור 8.16 – עץ משפחת Smith

א. רשום את הנתונים כביטוי מורכב של פרולוג באמצעות סריקת עץ.
ב. מצא את האריטי של הפנקטורים הבאים:

1) father
2) children
3) son

ג. תרגם לשאילתות של פרולוג את השאלות הבאות:
1. מהם השמות הפרטיים של האב, האם, הבן והבת?
2. מה הוא שם האב, כמה שנים הוא עובד בחברה ומה גובה משכורתו?
3. מהם שמות הבן והבת ומתי נולד כל אחד מהם?

שאלה 3

התבונן בביטוי המורכב קורס / 3 שבדוגמה 5 בסעיף 8.3. בהנחה כי הביטוי הוא חלק מבסיס נתונים גדול, בנה את החוקים האלה:

א. חוק השולף את השם של כל אחד מהקורסים, המועברים בסמסטר הראשון בלבד.
ב. חוק השולף את שם הקורס בו התרגול מתחיל מיד אחרי ההרצאה.
ג. חוק שבודק האם שתי הרצאות שונות ניתנות באותו יום, באותן השעות ובאותו אולם.
ד. חוק שבודק האם אותו מרצה מעביר שני קורסים שונים באותו יום של השבוע, ובאותם שעות.

הערה: החוקים ג' ו-ד' נועדו לבדוק את ה"בגים" של מערכת השעות.

שאלה 4

קובץ חשוב נמחק, אך ברשותך נשאר תדפיס, שמכיל שאלות ותשובות על בסיס הקובץ הזה. אתה מתבקש לשחזר את מבנה נתוני הקובץ על פי התדפיס הזה. לפניך שאילתה ותשובה. שחזר את העובדה, אשר על פיה התקבלה התשובה, והצג אותה בצורת עץ.

א. ?-מרכיבים(קטע(נקודה(X, Y),_), Z).
X = 1.
Y = 0.
Z = משולש(נקודה(1, 2), נקודה(5, 3), נקודה(1, 4)).

מהו לפי הבנתך, מבנה הארגומנט, אשר את מקומו תופס המשתנה האנונימי?

ב. ?-חיבור_מקבילי(X, _חיבור_טורי(Y, Z)).
X = חיבור_טורי(התנגדות1, התנגדות2).
Y = חיבור_מקבילי(התנגדות3, התנגדות4).
Z = התנגדות5.

8.6 אופרטורים ופעולות החשבון

כדי לבצע את פעולות החשבון, נעזרת מערכת הפרולוג בפרדיקטים, המוגדרים מראש והמכונים **פרדיקטי המערכת** (ראה סעיף 7.6). פרדיקטים אלו מנצלים את השגרות של מערכת התוכנה של המחשב, ולכן התנהגותם לעיתים שונה מן המצופה על פי עקרונות של תכנות לוגי ועל כך נדבר בהמשך. בסעיף זה נשתמש בפרדיקטים של הגירסה WISDOM PROLOG 1.3 (יתכן כי ישנם הבדלים בין פרדיקטי הגירסה הזאת לבין גירסאות אחרות).

מערכת של פרולוג מכירה את הפעולות האלו:

חיבור +
חיסור -
כפל *

הסימנים +, -, *, הם אטומים ומוגדרים בתור פרדיקטי המערכת. המערכת מכירה גם את סדר הקדימויות המקובל, ולכן הביטוי: 4*5+2 מתפרש כביטוי מורכב: ((4,5)*,2)+. הביטוי הזה נועד להציג את הפעולות, וכדי לבצען אנו נעזרים בפרדיקט נוסף: is. נציג את השאילתה לביצוע החישוב של הביטוי האריתמטי ונעיין בתשובה:

```
?- X  is  2 + 5 * 4.
X = 22
```

השאילתה מתקבלת על ידי המערכת כבעלת המבנה הבא:

```
?- is(X, +(2, *(5, 4)).
```

הצגת הפרדיקטים *, +, is בצורה של אופרטורים, כלומר בצורה בה הם נכתבים בין הארגומנטים ולא לפניהם, אינה מבשרת שינוי במבנה הנתונים. זוהי הצגה חיצונית בלבד. המתכנת יכול להגדיר אופרטורים משלו על-מנת לשפר את קריאות התכנית (ראה נספח ב'). כך, למשל, ניתן להגדיר את העובדה כי אברהם הוא אביו של יצחק, בצורה הבאה:

```
abraham  father_of  isaac.
```

ניתן להגדיר שלושה סוגי אופרטורים:

א. פרדיקט של אריטי 2 ניתן להגדיר כאופרטור הנכתב בין הארגומנטים, כדוגמת +, או *. אופרטור מסוג זה מכונה בלע"ז infix operator.

ב. פרדיקט של אריטי 1 ניתן להגדיר כאופרטור הנכתב לפני הארגומנט, כדוגמת not. אופרטור מסוג זה מכונה בשם prefix operator.

ג. פרדיקט של אריטי 1 ניתן להגדיר כאופרטור הנכתב אחרי הארגומנט. אופרטור מסוג זה נקרא postfix operator. לגירסה 1.3 של WISDOM PROLOG אין אופרטורים של המערכת מסוג זה.

הדרך בה ניתן להגדיר פרדיקט כאופרטור מתוארת בנספח ב'.

נעבור לפעולת החילוק. בגירסה 1.3 של WISDOM PROLOG ישנם שני פרדיקטים הקשורים לפעולת החילוק הרגילה. אלו הם הפרדיקטים "/" ו-mod. הביטוי a/b מסמל את החלק השלם של תוצאת החילוק של a ב-b. לדוגמה, אם a=16 וגם b=5, אז a/b מסמל את המספר 3. את שארית החילוק ניתן לחשב באמצעות הפרדיקט mod. נציג אם כן, את השאילתות ונבחן את התשובות:

```
?- X  is  16/5.
X = 3

?- X  is  16  mod  5.
X = 1
```

נציין כי הגירסה 1.3 של WISDOM אינה תומכת בנקודה צפה (floating point).

ניתן להשוות בין שני ביטויים חשבוניים באמצעות הפרדיקטים הבאים:

=:= "שווה"
=\= "איננו שווה"

ניתן לבטא את היחסים "גדול", "קטן", "איננו גדול" ו-"איננו קטן" באמצעות הסימנים הרגילים של האלגברה. בכתיבת הביטוי "איננו קטן/גדול" יש להקפיד שהסימן לא ידמה לחץ, כלומר יש לכתוב a=<b ולא a=>b. נביא דוגמאות אחדות:

```
?- 2 * 3 =< 9 - 2.
yes

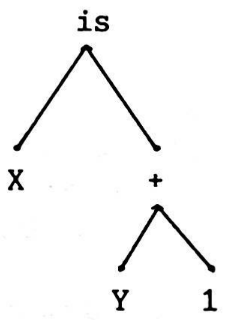
?- 1 >= 5.
no
```

הסימנים "=" ו-"\=" של האלגברה מנוצלים גם הם על ידי מערכת הפרולוג, אבל בהקשר אחר, השונה מן המוכר לנו באלגברה. נעיין בשאילתה ובתשובה:

```
?- X = 2 * 3.
X = 2 * 3
```

הפרדיקט "=" מאחד את המשתנה X עם הביטוי המורכב 2*3. הפרדיקט "\=" הוא שלילתו של "=".

תפקידם של פרדיקטי המערכת, העוסקים בהערכה וחישוב של ביטויים חשבוניים הינו לספק מישק לשגרות המתאימות של המחשב.
בשל השימוש בשגרות המחשב, התנהגות הפרדיקטים, האלה סוטה מן העקרון של תכנות לוגי. תכנית לוגית, שמגיעה למבוי סתום, אמורה לבצע נסיגה ולחפש דרך אלטרנטיבית לפתרון הבעיה. לכן תכנית לוגית מסתיימת בהצלחה (אם יש פתרון לבעיה), או בכישלון (אם אין פתרון), או שאינה מסתיימת כלל (אם היא נקלעת ללולאה אינסופית). בשל כך, "שגיאות זמן ריצה" אינן קורות בתכנות לוגי. עקרון זה מופר בעת שימוש בשגרות המחשב. נציג דוגמה כדי להסביר זאת.
נעיין בביטוי: X is Y + 1, ונתאר אותו בעזרת עץ:



איור 8.17 – עץ של הביטוי X is Y + 1

הביטוי הזה בנוי בהתאם לכללי התחביר של פרולוג. על פי ההגדרה של המשתנה הלוגי בתגובה לשאילתה "X is Y + 1-?" ניתן היה לצפות לתשובות רבות, האחת מהן למשל: X = 2, Y = 1.
נציג את השאילתה ונראה מה תהיה התשובה:

```
?-  X  is  Y + 1
Error in arithmetic expression
```

שגרת המחשב מחייבת שבמקום Y יוצב מספר. בהעדרו נשלחת הודעה של שגיאת זמן ריצה.

בסעיף 4 של חטיבה רביעית נביא הגדרה של מספר טבעי ושל פעולות החשבון באמצעות תכנית לוגית.

לסיום נביא שתי תכניות שידגימו את השימוש באופרטורים אותם למדנו.

התכנית הראשונה מוצאת את הגדול מבין שני מספרים נתונים.

```
max(X, Y, Max)  :- X >= Y, Max = X.
max(X, Y, Max)  :- X <  Y, Max = Y.
```

תכנית 8.3 – מקסימום של שני מספרים

נעיין בשאילתות ובתשובות:

```
?- max(1, 2, Max).
Max = 2

?- max(X, 2, Max).
Error in arithmetic expression
```

בהמשך נראה גירסאות נוספות לחישוב המקסימום.

התכנית השניה מחשבת סכום מספרים טבעיים מ-N עד M, כולל את N ואת M.

```
sum(N, N, N).
sum(N, M, Sum) :-
                N < M,
                M1 is M - 1,
                sum(N, M1, Sum1),
                Sum is Sum1 + M.
```

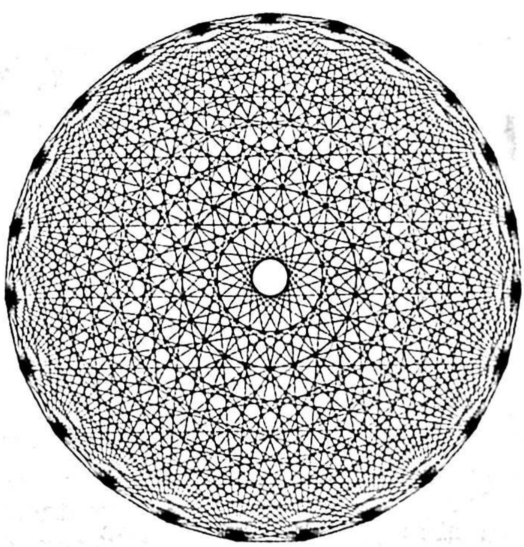
תכנית 8.4 – חישוב של סכום מספרים טבעיים מ-N עד M.

החוק הראשון קובע, שאם N=M, אז הסכום שווה ל-N. החוק השני הינו חוק רקורסיבי. הוא קובע שאם N<M, יש לחשב את הסכום עד M-1 ולחבר את התוצאה עם M. נעיין בשאילתות ובתשובות.

```
?- sum(2, 4, Sum).
Sum = 9

?- sum(N, 4, Sum).
Error in arithmetic expression
```

לאלו מן הקוראים, אשר גירסת פרולוג מצויה בהישג ידם, מומלץ להיעזר בפרדיקט trace על מנת להבין את הפעולה של תכנית 8.4.



פרק 9

רשימה

בפרק זה נכיר מבנה הקרוי רשימה. נציג את יעדי הלימוד.
בסעיף 9.1 נגדיר את הרשימה וננתח את מבנהו.
בסעיף 9.2 נראה את השימוש ברשימה לצורך בנייה של בסיס נתונים.
בסעיף 9.3 נציג 10 תכניות בנושא עיבוד של רשימות, אשר הקורא יתבקש לנתחן ולהפעילן.
בסעיף 9.4 נציג את התוצאות של הפעלת חלק מהתכניות של הסעיף הקודם.
בסעיף 9.5 נלמד לפרק רשימה לתת-רשימות. נציג 9 אפשרויות לפירוק הרשימה ונבקש מן הקורא להפעילן ולהשוותן.
בסעיף 9.6 נעצב יחד עם הקורא תכניות לוגיות. נתאר את הבעיה ואת אסטרטגיית הפתרון ונתרגם את התאור הזה לפרולוג. נתחיל מן התכנית "מיון טבעי", בה נתאר את צעדי הפתרון בצורה כללית. נמשיך בתכניות, אשר בהן תאור של צעדי הפתרון יעשה ביתר פירוט.
בסעיף 9.7 נפעיל ונשווה את תכניות המיון של הסעיף הקודם.

9.1 הגדרה של רשימה

לפנינו בסיס נתונים:

```
לומד(דני, פרולוג).
לומד(דני, ספרות).
לומד(דני, אנגלית).
לומד(יוסי, מוסיקה).
לומד(יוסי, ספרות).
```

שלוש העובדות הראשונות דנות במקצועות אותם לומד דני, ושתי העובדות הנוספות דנות במקצועות של יוסי. נארגן את הנתונים באמצעות שתי עובדות אלו:

```
לומד(דני, פרולוג, ספרות, אנגלית).
לומד(יוסי, מוסיקה, ספרות).
```

הארגון הזה איננו יעיל, שהרי דני מתואר באמצעות היחס לומד / 4 ויוסי – באמצעות היחס לומד / 3. על כן, מבנה השאילתה שמתייחסת אל דני שונה מזו המתייחסת ליוסי.

?- לומד(X, Y, Z).

X = יוסי
Y = מוסיקה
Z = ספרות
לא (כלומר, אין תשובות נוספות).

?- לומד(X, Y, Z, W).

X = דני
Y = פרולוג
Z = ספרות
W = אנגלית
לא (כלומר, אין תשובות נוספות).

שפת הפרולוג מאפשרת להציג את המקצועות של דני או של יוסי כעצם אחד:

לומד(דני, [פרולוג, ספרות, אנגלית]).
לומד(יוסי, [מוסיקה, ספרות]).

סדרה של איברים (המופרדים זה מזה בסימן פסיק), שמוקפת בסוגריים מרובעים, קרויה **רשימה** (list). את הרשימה על כל איבריה, ניתן לראות כיישות עצמאית, ועל כן ניתן להתייחס אל דני ואל יוסי באמצעות שאילתה אחת. נעיין בשאילתה ובתשובות:

?- לומד(X, Y).

X = דני
Y = [פרולוג, ספרות, אנגלית];

X = יוסי
Y = [מוסיקה,ספרות].

ברוב הגירסאות של פרולוג אין מגבלות על המבנה התחבירי של איברי הרשימה, לדוגמה:

לומד(רמי, [סמסטר(1), מתמטיקה(אלגברה), [אנגלית, התעמלות]].
לומד([רמי, [סמסטר(2), מחשבים([פרולוג, מבנה_נתונים])]).

מבין שלושת האיברים של הרשימה הראשונה, האיבר השלישי הוא רשימה: [אנגלית, התעמלות]. מבין שני האיברים של הרשימה השניה, האיבר השני הוא ביטוי מורכב, אשר הארגומנט שלו הוא רשימה:

מחשבים([פרולוג, מבנה_נתונים]).

מכאן אנו לומדים, כי רשימה היא גוף עצמאי כמו חברה בע"מ. את מדרג החברה אפשר לתאר באופן רקורסיבי: בראש החברה עומד הנשיא, ופיקודיו הם חברה בזכות עצמה. הראשון בין פיקודיו הוא סגן-הנשיא, הבאים אחריו במדרג החברה הם חברה בפני עצמה, וכן הלאה.

לרשימה יש מבנה דומה לזה של החברה שתיארנו לעיל – מבנה רקורסיבי. האיבר הראשון של הרשימה מכונה **ראש** (head). שאר איבריה מכונים **זנב** (tail) והם מהווים רשימה בזכות עצמה. את המבנה הרקורסיבי של הרשימה תיארנו באמצעות איור 9.1.

בפרולוג העברי:

רשימה	ראש	זנב
[פרולוג,ספרות,אנגלית]	פרולוג	[ספרות,אנגלית]
[[א], [ב,ג], הכל]	[א]	[[ב,ג], הכל]
[סמסטר(2),[ציור,פיסול]]	סמסטר(2)	[[ציור,פיסול]]
[יום(שני), מהירות(5)]	יום(שני)	[מהירות(5)]

בפרולוג האנגלי:

List	Head	Tail
[1,2,3]	1	[2,3]
[[1],[2,3],all]	[1]	[[2,3],all]
[semester(2),[prolog, english]]	semester(2)	[[prolog,english]]
[day(1,may,1980), speed(5)]	day(1,may, 1980)	[speed(5)]

איור 9.1 – המבנה הרקורסיבי של הרשימה: הראש הינו איבר והזנב הינו רשימה

בדומה לחברה אשר יכולה לפטר את כל חבריה, כולל הנשיא, ועדיין להיות רשומה אצל רשם החברות כגוף עצמאי – כך קיימת גם רשימה ללא איברים. הרשימה אשר אין בה איברים נקראת **רשימה ריקה** (empty list) ומסומנת כך:[]. מבחינה תחבירית, הרשימה הריקה [] היא אטום של פרולוג.

הרשימה של הפרולוג העברי בת איבר אחד, כדוגמת הרשימה [יוסי], היא ביטוי מורכב בעל פנקטור ושני ארגומנטים. הארגומנט הראשון הוא האיבר יוסי, והארגומנט השני הוא הרשימה

הריקה. האיבר יוסי הוא הראש של הרשימה [יוסי], והאטום [] הוא הזנב של [יוסי]. הרשימה של הפרולוג העברי, אשר ראשה יוסי וזנבה אינו ידוע, מסומנת כך: [יוסי|T]. המשתנה T מסמן את הזנב הבלתי ידוע. את הרשימה של הפרולוג העברי, אשר גם ראשה וגם זנבה אינם ידועים, ניתן לסמן בשני דרכים. ניתן לסמנה באמצעות משתנה, כפי שעשינו זאת בשאילתה ?-לומד(X, Y). ניתן גם לסמנה באמצעות שני משתנים: [H|T]. המשתנה H מסמן את ראש הרשימה, והמשתנה T מסמן את זנבה. נדגיש כי המעמד של H שונה מזה של T: המשתנה H מסמל איבר ברשימה, והמשתנה T מסמל רשימה.
לעתים אנו ניעזר גם בסימון [_ראש|_זנב]. שפת הפרולוג מסווגת רשימה כלשהי [_ראש|_זנב] כמבנה מורכב בעל שני ארגומנטים: _ראש, _זנב. את הפנקטור של המבנה הזה מסמנים ברוב הגירסאות של פרולוג באמצעות הנקודה "." (dot). להלן דוגמאות:

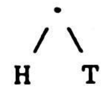
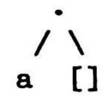
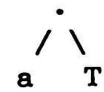
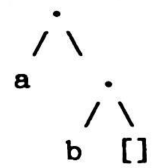
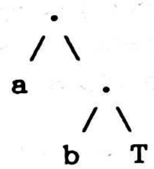
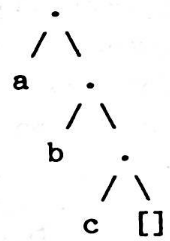
רשימה כמבנה מורכב	רשימה באמצעות הסוגריים
.(_ראש, _זנב)	[_ראש, _זנב]
.(יוסי, [])	[יוסי]
.(יוסי, T)	[יוסי|T]
.(יוסי, .(דני, []))	[יוסי, דני]
.(יוסי, .(דני, T))	[יוסי, דני|T]
.(יוסי, .(דני, .(רמי, [])))	[יוסי, דני, רמי]

גם בפרולוג האנגלי קיימת רשימה ריקה, שגם היא אטום של פרולוג. רשימה של הפרולוג האנגלי אשר ראשה H וזנבה T, מסומנת כך: [H|T]. שפת הפרולוג מסווגת את [H|T] כמבנה מורכב בעל שני ארגומנטים: H ו-T. את הפנקטור של המבנה הזה מסמנים באמצעות "." (dot). להלן דוגמאות:

רשימה כמבנה מורכב	רשימה באמצעות הסוגריים
.(H,T)	[H|T]
.(a,[])	[a]
.(a,T)	[a|T]
.(a,.(b,[]))	[a,b]
.(a,.(b,T))	[a,b|T]
.(a,.(b,.(c,[])))	[a,b,c]

מערכת הפרולוג רואה את הרשימה כמבנה מורכב בעל פנקטור ושני ארגומנטים. למרות זאת, מעדיפים כותבי התכניות לייצג את הרשימה באמצעות הסוגריים המרובעים, למען הקריאות והבהירות (השווה נא את [a,b,c] ואת ייצוגה כמבנה מורכב).

כמו כל מבנה מורכב, ניתן לייצג את הרשימה באמצעות העץ. ייצוג זה הובא באיור 9.2.

רשימה כמבנה מורכב	רשימה באמצעות הסוגריים
	[H|T]
	[a]
	[a|T]
	[a,b]
	[a,b|T]
	[a,b,c]

איור 9.2 – ייצוג של הרשימה בצורת עץ

למדנו אם-כן, כי הרשימה היא מבנה רקורסיבי. ביטוי מורכב בעל שני ארגומנטים הוא רשימה, אם הארגומנט השני שלו הוא רשימה. ואיך נדע אם הארגומנט השני הוא רשימה? – על פי המבנה הרקורסיבי. כלומר, הארגומנט השני הוא ביטוי מורכב בעל שני ארגומנטים, שהשני בו הוא רשימה, וחוזר חלילה, עד שנגיע לאטום [], אשר הינו רשימה ריקה, משמע – רשימה.

התכנית הבאה מגדירה את הרשימה באמצעות רקורסיה. החוק הראשון קובע, כי האטום [] הוא רשימה. החוק השני קובע, כי מבנה מורכב בעל שני ארגומנטים הוא רשימה, אם הארגומנט השני הוא רשימה:

```
רשימה([]).                                  list([]).
רשימה([H|T]):- רשימה(T).                    list([H|T]):- list(T).
```

תכנית 9.1 – הגדרת רשימה (גירסאות אנגלית ועברית)

תרגיל 9.1

לפניך שלוש רשימות של הפרולוג האנגלי:

1. `[[ ]]`
2. `[[ ],[ ]]`
3. `[[H|T],[a,b|X]]`

תאר אותן בשני אופנים:

א. כביטוי מורכב.
ב. באמצעות עץ.

9.2 בניית בסיסי נתונים עם רשימות

אחרי שהכרנו את הרשימה כגוף אחיד, אנו מעוניינים להכיר את כל אחד מאיבריה. התכנית הבאה מגדירה את איברי הרשימה. החוק הראשון קובע כי האיבר שייך לרשימה אם הוא עומד בראשה. החוק השני הוא חוק רקורסיבי: האיבר שייך לרשימה אם הוא שייך לזנבה.

```
חבר_רשימה(X, [X|T]).                        member(X, [X|T]).
חבר_רשימה(X, [Y|T]):-                       member(X, [Y|T]):-
     חבר_רשימה(X, T).                            member(X, T).
```

תכנית 9.2 – חבר_ברשימה (בגירסה אנגלית ועברית)

נתבונן בשאילתה:

```
?- member(X,[a,b,c]).
```

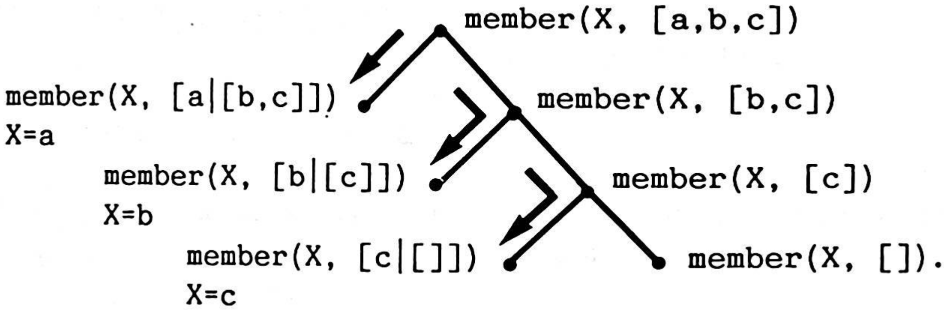
על פי החוק הראשון נקבל את התשובה הבאה:

```
X = a;
```

אחרי שמקישים ";", פרולוג פונה אל החוק השני, הקובע כי X הוא חבר ברשימה המקורית אם הוא חבר ברשימה [b,c]. כאן מפעיל פרולוג את החוק הראשון ועל פיו קובע כי:

```
X = b;
```

וחוזר חלילה: הקשת ";", פניה אל החוק השני אשר על פיו יש לברר אם X שייך לרשימה [c], חזרה אל החוק הראשון והתשובה: X = c. את תהליך החיפוש אחר התשובה לשאילתה הנ"ל נתאר באמצעות איור 9.3:



איור 9.3 – חישוב של השאילתה ?- member(X, [a,b,c])

תרגיל 9.2

לפניך השאילתה והתשובות שהתקבלו:

```
?- member(a, X).
X = [a|T];
X = [Y, a|T];
X = [Y1, Y2, a|T]
```

תן הסבר לתשובות אלה.

את השימוש ברשימות לצורך בניית בסיסי נתונים נדגים באמצעות תכנית 9.3.

```
לומד(רמי, [אלגברה, ספרות, מוסיקה]).
לומד(נימרוד, [פרולוג, ספרות]).
לומד(סיגל, [פרולוג, מוסיקה, אלגברה]).
לומד(אביאור, [פרולוג, ביולוגיה, אלגברה, ספרות]).
סופר(X):- לומד(X, L), חבר_ברשימה(ספרות, L).
איש_מחשבים(X):- לומד(X, L), חבר_ברשימה(פרולוג, L),
        חבר_ברשימה(אלגברה, L).
חבר_ברשימה(X, [X|T]).
חבר_ברשימה(X, [Y|T]):- חבר_ברשימה(X, T).
```

תכנית 9.3 – רשימת המקצועות

נשווה בין תכנית "תלמידים" שבפרק 7 לבין תכנית 9.3. בשתיהן מופיע הפנקטור לומד / 2, אלא שהארגומנט השני של לומד / 2 בתכנית 9.3 הוא רשימה, וזה מחייב שינוי בהגדרות של סופר ושל איש_מחשבים. הגדרת סופר של תכנית 9.3 קובעת כי X הוא סופר, אם X לומד את רשימת המקצועות L וספרות שייכת לרשימה הזאת.

התכנית 9.4 היא דוגמה נוספת של שימוש ברשימה לשם בנייה של בסיס נתונים (השווה נא בינה לבין דוגמה 8.4 בפרק הקודם).

```
משפחה(אב(ישראלי, ישראל), אם(ישראלי, ישראלה),
            ילדים([דני,יוסי,רמי])).
משפחה(אב(צבר, יורם), אם(צבר, רינה), ילדים([])).
ילדים(_המשפחה, _רשימת_הילדים):-
        משפחה(אב(_המשפחה, _), _, ילדים(_רשימת_הילדים)).
ילד(_שם_הילד, _שם_המשפחה):- ילדים(_שם_המשפחה, _רשימת_הילדים),
        חבר_ברשימה(_שם_הילד, _רשימת_הילדים).

/* את הגדרת היחס חבר_ברשימה/2 ראה בתכנית 9.2 */
```

תכנית 9.4 – ילדי משפחה

להלן שאילתות ותשובות לדוגמה:

```
?- משפחה(X, Y, Z).
X = אב(ישראלי, ישראל)
Y = אם(ישראלי, ישראלה)
Z = ילדים([דני, יוסי, רמי]);

X = אב(צבר, יורם)
Y = אם(צבר, רינה)
Z = ילדים([])

?- ילדים(X, Y).
X = ישראלי
Y = [דני, יוסי, רמי];

X = צבר
Y = []
```

נציין, כי ברוב הגירסאות של פרולוג היחס חבר_ברשימה הוגדר כפרדיקט המערכת בשל השימוש הרב שעושים בו.

תרגיל 9.3

מצא על פי תכנית 9.4 את התשובות של פרולוג לשאילתה:

```
?- ילד(_שם_פרטי, _שם_משפחה).
```

נסיים בתכנית תרגום נוסח תכנית Dictionary שבפרק 5.

```
means(quisiera, i_should_like).
means(ella, she).
means(usar, to_use).
means(comprar, to_buy).
means(habla, speaks).
means(aquella, that).
means(espanol, spanish).
means(silla, chair).

translate([], []).
translate([X|T], [Y|R]):- means(X, Y), translate(T, R).
```

תכנית 9.5 – תרגום

נעיין בשאילתות ובתשובות:

```
?- translate([ella, habla, espanol], X).
X = [she, speaks, spanish]

?- translate(X, [i_should_like, to_use, that, chair]).
X = [quisiera, usar, aquella, silla]
```

9.3 עיבוד של רשימות – תרגול

בסעיף זה נציג מספר תכניות המשמשות לעיבוד רשימות. חלק של היחסים שמוגדרים בתכניות, הוגדרו באמצעות סוג רקורסיה, שמכונה בלע"ז left-side recursion. זוהי רקורסיה, אשר בה היחס המוגדר חוזר רקורסיבית בגוף החוק **מיד** אחרי הקשר הלוגי IF. רקורסיה מסוג זה רגישה מאוד לארגון פסוקי התכנית – השווה למשל, את היחסים append ו-append1 בתרגיל 9.3(א). לנושא ארגון הפסוקים והשפעתו על תהליך קבלת תשובה לשאילתה נחזור בפרק 10, במסגרת דיון במודל חישובי של תכנית לוגית.

מטרת סעיף זה היא להקנות מיומנות בעבודה עם רשימות. המטרה הנוספת היא ללמוד **להפעיל** תכנית אשר כבר נכתבה. בסעיף הבא נעשה צעד נוסף קדימה: נלמד **לכתוב** תכניות.
את אוסף החוקים והעובדות, בעלי ראש זהה, נכנה בשם **פרוצדורה** (procedure).
מומלץ בעת ההרצה של הפרוצדורות שלהלן להיעזר ב-trace. חלק של היחסים, שמופיעים בפרוצדורות, הוגדרו כפרדיקטי המערכת בגירסאות שונות של פרולוג.
בסעיף הבא נביא פתרונות של חלק מהתרגילים, אשר הוצגו להלן.

ניתן לשרשר (append, concatenate) שתי רשימות. הרשימה [א,ב,ג] היא שרשור הרשימות [א] ו-[ב,ג]. היא גם שרשור של הרשימות [] ו-[א,ב,ג]. נדגים זאת באמצעות התכנית הבאה:

```
/* שרשור(L1,L2,L3) :- הרשימה L3 היא */
/* שרשור של הרשימות L1 ו-L2 */
```

```
append([], L, L).
append([X|L1], L2, [X|L3]):-
        append(L1, L2, L3).
```

```
שרשור([], L, L).
שרשור([X|L1], L2, [X|L3]):-
        שרשור(L1, L2, L3).
```

פרוצדורה 9.3(א) – שרשור רשימות (גירסה עברית ואנגלית)

בגלל חשיבותו של היחס append, הוא מוגדר בגירסאות פרולוג רבות כפרדיקט המערכת.

תרגיל 9.3(א)

א. בדוק מהן התשובות של פרולוג לשאילתות האלה:

```
?- append([a],[b,c],X).
?- append([a],X,[a,b,c]).
?- append(X,[b,c],[a,b,c]).

?- append(X,Y,[a,b,c]).        (הפתרון באיור 9.4)
?- append(X,[a,b],Y).          (הפתרון באיור 9.5)
?- append([a,b],X,Y).

?- append(X,Y,Z).
```

ב. נגדיר את פרוצדורה append1 כך:

```
append1([X|L1], L2, [X|L3]):-
                   append1(L1, L2, L3).
append1([], L, L).
```

נסח את השאילתות שלעיל כלפי append1, ובדוק מה הן התשובות.

9.3(ב) פרוצדורה select

בהמשך נציג רק את הגירסאות האנגליות של הפרוצדורות. הקורא המעוניין יגדיר בנקל פרוצדורות אלו גם בפרולוג העברי.

התכנית הבאה מוחקת מן הרשימה את הופעתו הראשונה של האיבר. היא מורכבת משני חוקים. החוק הראשון קובע כי אם האיבר נמצא בראש הרשימה, אז תוצאת מחיקתו היא הזנב של הרשימה. החוק השני הוא חוק רקורסיבי: כדי למחוק איבר הנמצא בזנבה של הרשימה, יש למחוק אותו מזנבה.

```
/*   select(X, List, Newlist):- Newlist is the result   */
/*          of deleting one occurence of X in List       */

select(X, [X|Tail], Tail).
select(X, [Y|Tail], [Y|T]):- select(X, Tail, T).
```

פרוצדורה 9.3(ב) – מחיקה של הופעת איבר

נעיין בשאילתות ובתשובות הבאות:

```
?- select(X,[1,2,3],Y).
X = 1
Y = [2,3];

X = 2
Y = [1,3];

X = 3
Y = [1,2]


?- select(a,[a,b,a,a],NewList).
Newlist = [b,a,a];
Newlist = [a,b,a];
Newlist = [a,b,a];
no(more answers)
```

תרגיל 9.3(ב)

בדוק מהן התשובות לשאילתות האלה:

```
?- select(3, List, [1,2,3]).
?- select(X, List, [a,b,c]).
?- select(X, List, NewList).
```

(הפתרון באיור 9.6)

9.3(ג) פרוצדורה last

התכנית הבאה מאתרת את האיבר האחרון של הרשימה:

```
/* last(X,L):- X is the last member of a  non_empty list L */

last(X, [X]).
last(X, [Head|Tail]):- last(X, Tail).
```

פרוצדורה 9.3(ג) – בחירת האיבר האחרון

ניתן לבחור את האיבר האחרון בעזרת append/3:

```
last1(X, L):- append(_ , [X], L).
```

תרגיל 9.3(ג)

בדוק מהן התשובות לשאילתות האלה:

`?- last(X, [1,2,3]).`	(הפתרון באיור 9.7)
`?- last(a, List).`	
`?- last(X, List).`	(הפתרון באיור 9.8)
`?- last1(X, [1,2,3]).`	(הפתרון באיור 9.9)
`?- last1(a, List).`	
`?- last1(X, List).`	

הערה: בנושא פרוצדורה last, ראה איור 10.11 בפרק 10

9.3(ד) פרוצדורה double

התכנית הבאה מעתיקה פעמיים כל אחד מאיברי הרשימה:

```
/*  double([X|T], [X, X|NewT]):- each of the elements of   */
/*                  a nonempty List is doubled in NewList   */

double([X|T], [X, X|NewT]):- double(T, NewT).
double([X], [X, X]).
```

פרוצדורה 9.3(ד) – הכפלת איברי הרשימה

תרגיל 9.3(ד)

בדוק מהן התשובות לשאילתות האלה:

```
?-double([a,b,c], X).
?-double(X, [a,a,b,b,c,c]).
?-double(X, [a,a,b,c,c]).
```

(הפתרון באיור 9.10)

הערה: בנושא פרוצדורה double ראה איור 10.10 בפרק 10.

9.3(ה) פרוצדורה prefix

נעיין במילה ararat. האותיות a, ar, ara וכן הלאה, הן **קידומות** (prefixes) של ararat. התכנית הבאה מגדירה את הקידומת של הרשימה.

```
/*  prefix(Pre, List):- the list Pre is a prefix of List  */

prefix([], List).
prefix([X|Pre], [X|Tail]):- prefix(Pre, Tail).
```

פרוצדורה 9.3(ה) – קידומת של הרשימה

תרגיל 9.3(ה)

מצא את התשובות של פרולוג לשאילתות האלה:

```
?- prefix([a,b], [a,b,c]).
?- prefix(X, [a,b,c]).
?- prefix([a,b], List).
?- prefix(X, List).
```

(הפתרון באיור 9.11)
(הפתרון באיור 9.12)

9.3(ו) פרוצדורה suffix

נתבונן במילה ararat. האותיות t, at, rat וכן הלאה, הן **סיומות** (suffixes) של ararat. התכנית הבאה מגדירה את סיומת הרשימה.

```
/*  suffix(Suf, List):- the list Suf is a suffix of List  */

suffix(List, List).
suffix(Suf, [X|Tail]):- suffix(Suf, Tail).
```

פרוצדורה 9.3(ו) – סיומת של הרשימה

תרגיל 9.3(ו)

מצא את התשובות לשאילתות האלה:

```
?- suffix([b,c], [a,b,c]).
?- suffix(X, [a,b,c]).          (הפתרון באיור 9.13
?- suffix([a,b], List).         (הפתרון באיור 9.14
?- suffix(X, List).
```

9.3(ז) פרוצדורה permutation

התכנית הבאה מגדירה את התמורות של הרשימה.

```
/*  permutation(List, Permut):- the list Permut             */
/*                              is a permutation of List    */

permutation([], []).
permutation(List, [X|Permut]):-
                     select(X, List, Rest),
                     permutation(Rest, Permut).
/*                 את הפרוצדורה select הגדרנו ב-9.3(ב)     */
```

פרוצדורה 9.3(ז) – תמורה של הרשימה

תרגיל 9.3(ז):

מצא את התשובות לשאילתות האלה:

```
?- permutation([a,b,c,d], P).
?- permutation(X, [a,b,c,d]).     (הפתרון באיור 9.15
```

9.3(ח) פרוצדורה sequence

בהינתן מספר טבעי N, הפרוצדורה sequence יוצרת את רשימת המספרים מ-1 עד N.

```
/*  sequence(N, Sequence):- the list Sequence          */
/*                     is a list of numbers from 1 to N  */

sequence(1, [1]).
sequence(N, Sequence):-
                    N>1,
                    N1 is N-1,
                    sequence(N1, S1),
                    append(S1, [N], Sequence).
/*                 את הפרוצדורה append הגדרנו ב-9.3(א)  */
```

פרוצדורה 9.3(ח) – רשימת המספרים מ-1 עד N

תרגיל 9.3(ח)

א. מצא את התשובה לשאילתה הבאה:

```
?- sequence(5, Sequence).
```

(הפתרון הינו חלק מאיורים)
(22, 23 של חטיבה רביעית.)

ב. בדוק האם תשתנה התשובה, אם נחליף בהגדרת sequence את הפרוצדורה append בפרוצדורה append1.

9.3(ט) פרוצדורה between

התכנית הבאה מגדירה את רשימת המספרים הטבעיים, הנמצאים בין המספרים N ו-M, לפי סדר עולה.

```
/*  between(N,M,List):- List is the sequence of  */
/*                      numbers between N and M   */

between(N, N, [N]).
between(N, M, [N|Tail]):-
                    N<M,
                    N1 is N+1,
                    between(N1, M, Tail).
```

פרוצדורה 9.3(ט) – סדרת המספרים בין N ו-M

תרגיל 9.3(ט)

התבונן בפרוצדורה between והגדר את הפרוצדורה sequence מבלי להשתמש ב-append.

9.4 עיבוד של רשימות – פתרונות

להלן פתרונות של חלק מתרגילי סעיף 9.3. נציין כי append הינו פרדיקט המערכת של WISDOM-PROLOG. המערכת אינה מאפשרת למשתמש להגדיר את פרדיקטי המערכת, ולכן החלטנו "לרמות" אותה. היחס aappend מוגדר באופן זהה ל-append של 9.3(א). מבחינת המערכת, הפרדיקטים append ו-aappend הינם שונים. מבחינה סמנטית הם שקולים.

```
tracing: ¦ ?- aappend(X,Y,[a,b,c]).
[0] call: aappend(X80,Y81,[a,b,c])?
[0] exit: aappend([],[a,b,c],[a,b,c])?
... in 1 reductions.

X = []
Y = [a,b,c] ;
[0] redo: aappend([],[a,b,c],[a,b,c])?
 [1] call: aappend(L1184,L2185,[b,c])?
 [1] exit: aappend([],[b,c],[b,c]).
[0] exit: aappend([a],[b,c],[a,b,c]).
... in 2 reductions.

X = [a]
Y = [b,c] ;
[0] redo: aappend([a],[b,c],[a,b,c]).
 [1] redo: aappend([],[b,c],[b,c]).
  [2] call: aappend(L1278,L2279,[c]).
  [2] exit: aappend([],[c],[c]).
 [1] exit: aappend([b],[c],[b,c]).
[0] exit: aappend([a,b],[c],[a,b,c]).
... in 3 reductions.
```

(המשך)

```
X = [a,b]
Y = [c] ;
[0] redo: aappend([a,b],[c],[a,b,c]).
 [1] redo: aappend([b],[c],[b,c]).
  [2] redo: aappend([],[c],[c]).
   [3] call: aappend(L1359,L2360,[]).
   [3] exit: aappend([],[],[]).
  [2] exit: aappend([c],[],[c]).
 [1] exit: aappend([b,c],[],[b,c]).
[0] exit: aappend([a,b,c],[],[a,b,c]).
... in 4 reductions.

X = [a,b,c]
Y = []
```

הרשימה [a,b,c] משורשרת מרשימות X ו-Y, אשר אינן ידועות.
הוצגו כל ארבע התשובות האפשריות.

איור 9.4 – aappend(X, Y, [a,b,c])

```
tracing: ¦ ?- aappend(X,[a,b],Y).
[0] call: aappend(X80,[a,b],Y86)?
[0] exit: aappend([],[a,b],[a,b])?
... in 1 reductions.

X = []
Y = [a,b] ;
[0] redo: aappend([],[a,b],[a,b])?
 [1] call: aappend(L1182,[a,b],L3185).
 [1] exit: aappend([],[a,b],[a,b]).
[0] exit: aappend([X181],[a,b],[X181,a,b]).
... in 2 reductions.

X = [X181]
Y = [X181,a,b] ;
[0] redo: aappend([X181],[a,b],[X181,a,b]).
 [1] redo: aappend([],[a,b],[a,b]).
  [2] call: aappend(L1263,[a,b],L3266).
  [2] exit: aappend([],[a,b],[a,b]).
 [1] exit: aappend([X262],[a,b],[X262,a,b]).
[0] exit: aappend([X181,X262],[a,b],[X181,X262,a,b]).
... in 3 reductions.

X = [X181,X262]
Y = [X181,X262,a,b]
```

הרשימה Y משורשרת מרשימות X ו-[a,b].
הוצגו שלוש תשובות; ניתן לקבל תשובות נוספות.

איור 9.5 – aappend(X, [a,b], Y)

```
tracing: ¦ ?- select(X,List,[a,b,c]).
[0] call: select(X80,List81,[a,b,c])?
[0] exit: select(X182,[X182,a,b,c],[a,b,c])?
... in 1 reductions.

X = X182
List = [X182,a,b,c] ;
[0] redo: select(X182,[X182,a,b,c],[a,b,c])?
 [1] call: select(X182,Tail185,[b,c])?
 [1] exit: select(X268,[X268,b,c],[b,c])?
[0] exit: select(X268,[a,X268,b,c],[a,b,c])?
... in 2 reductions.

X = X268
List = [a,X268,b,c] ;
[0] redo: select(X268,[a,X268,b,c],[a,b,c])?
 [1] redo: select(X268,[X268,b,c],[b,c])?
  [2] call: select(X268,Tail271,[c])?
  [2] exit: select(X354,[X354,c],[c])?
 [1] exit: select(X354,[b,X354,c],[b,c])?
[0] exit: select(X354,[a,b,X354,c],[a,b,c])?
... in 3 reductions.

X = X354
List = [a,b,X354,c] ;
[0] redo: select(X354,[a,b,X354,c],[a,b,c])?
 [1] redo: select(X354,[b,X354,c],[b,c])?
  [2] redo: select(X354,[X354,c],[c])?
   [3] call: select(X354,Tail357,[])?
   [3] exit: select(X440,[X440],[])?
  [2] exit: select(X440,[c,X440],[c])?
 [1] exit: select(X440,[b,c,X440],[b,c])?
[0] exit: select(X440,[a,b,c,X440],[a,b,c])?
... in 4 reductions.

X = X440
List = [a,b,c,X440]
```

הרשימה [a,b,c] הינה התוצאה של בחירת X מרשימה בלתי ידועה List.
הוצגו כל ארבע התשובות האפשריות.

איור 9.6 – select(X, List, [a,b,c])

```
tracing: ¦ ?- last(X, [1,2,3]).
[0] call: last(X80,[1,2,3])?
 [1] call: last(X186,[2,3]).
  [2] call: last(X265,[3]).
  [2] exit: last(3,[3]).
 [1] exit: last(3,[2,3]).
[0] exit: last(3,[1,2,3]).
... in 3 reductions.

X = 3
```

האיבר X הינו האחרון ברשימה [1,2,3]. התשובה: X=3.

איור 9.7 – last(X, [1,2,3])

```
tracing: ¦ ?- last(X,List).
[0] call: last(X80,List81)?
[0] exit: last(X175,[X175])?
... in 1 reductions.

X = X175
List = [X175] ;
[0] redo: last(X175,[X175])?
 [1] call: last(X175,Tail178)?
 [1] exit: last(X259,[X259])?
[0] exit: last(X259,[Head177,X259])?
... in 2 reductions.

X = X259
List = [Head177,X259]
```

האיבר X הינו האחרון ברשימה List, אשר אינה ידועה, וזאת גם המשמעות התשובות שהתקבלו.

איור 9.8 – last(X, List)

```
tracing: ¦ ?- last1(X, [1,2,3]).
[0] call: last1(X80,[1,2,3])?
 [1] call: aappend(_188,[X186],[1,2,3]).
  [2] call: aappend(L1268,[X186],[2,3]).
   [3] call: aappend(L1349,[X186],[3]).
   [3] exit: aappend([],[3],[3]).
  [2] exit: aappend([2],[3],[2,3]).
 [1] exit: aappend([1,2],[3],[1,2,3]).
[0] exit: last1(3,[1,2,3]).
... in 4 reductions.

X = 3
```

האיבר X הינו האחרון ברשימה [1,2,3]. החישוב נעשה באמצעות append.

איור 9.9 – last1(X, [1,2,3])

```
tracing: ¦ ?- double(X,[a,a,b,b,c,c]).
[0] call: double(X80,[a,a,b,b,c,c])?
 [1] call: double(T186,[b,b,c,c])?
  [2] call: double(T272,[c,c])?
   [3] call: double(T358,[])?
   [3] fail: double(T358,[])?
  [2] exit: double([c],[c,c])?
 [1] exit: double([b,c],[b,b,c,c])?
[0] exit: double([a,b,c],[a,a,b,b,c,c])?
... in 3 reductions.

X = [a,b,c]
```

הרשימה [a,a,b,b,c,c] הינה הרשימה X המוכפלת.

איור 9.10 – double(X, [a,a,b,b,c,c])

```
tracing: | ?- prefix([a,b],X).
[0] call: prefix([a,b],X85)?
 [1] call: prefix([b],Tail180)?
  [2] call: prefix([],Tail265)?
  [2] exit: prefix([],List347)?
 [1] exit: prefix([b],[b|List347])?
[0] exit: prefix([a,b],[a,b|List347])?
... in 3 reductions.

X = [a,b|List347] ;
[0] redo: prefix([a,b],[a,b|List347])?
 [1] redo: prefix([b],[b|List347])?
  [2] redo: prefix([],List347)?
  [2] fail: prefix([],_265)?
 [1] fail: prefix([b],_180)?
[0] fail: prefix([a,b],_85)?

no
```

הרשימה [a,b] הינה קידומת של הרשימה הבלתי ידועה X.
התשובה מתפרשת כך: a ו-b הינם שני איבריה הראשונים של X.

איור 9.11 – prefix([a,b], X)

```
tracing: | ?- prefix(X,List).
[0] call: prefix(X80,List81)?
[0] exit: prefix([],List176)?
... in 1 reductions.

X = []
List = List176 ;
[0] redo: prefix([],List176)?
 [1] call: prefix(Pre177,Tail179)?
 [1] exit: prefix([],List261)?
[0] exit: prefix([X176],[X176|List261])?
... in 2 reductions.

X = [X176]
List = [X176|List261]
```

הרשימה X הינה קידומת של הרשימה List; שתיהן אינן ידועות.

איור 9.12 – prefix(X, List)

```
tracing: | ?- suffix(X,[a,b,c]).
[0] call: suffix(X80,[a,b,c])?
[0] exit: suffix([a,b,c],[a,b,c])?
... in 1 reductions.

X = [a,b,c] ;
[0] redo: suffix([a,b,c],[a,b,c])?
 [1] call: suffix(Suf178,[b,c])?
 [1] exit: suffix([b,c],[b,c])?
[0] exit: suffix([b,c],[a,b,c])?
... in 2 reductions.

X = [b,c]
```

הרשימה X הינה סיומת של הרשימה [a,b,c].
הוצגו שתי תשובות; ישנן תשובות נוספות.

איור 9.13 – suffix(X, [a,b,c])

```
tracing: ¦ ?- suffix([a,b],List).
[0] call: suffix([a,b],List85)?
[0] exit: suffix([a,b],[a,b])?
... in 1 reductions.

List = [a,b] ;
[0] redo: suffix([a,b],[a,b])?
 [1] call: suffix([a,b],Tail179)?
 [1] exit: suffix([a,b],[a,b])?
[0] exit: suffix([a,b],[X178,a,b])?
... in 2 reductions.

List = [X178,a,b] ;
[0] redo: suffix([a,b],[X178,a,b])?
 [1] redo: suffix([a,b],[a,b])?
  [2] call: suffix([a,b],Tail263)?
  [2] exit: suffix([a,b],[a,b])?
 [1] exit: suffix([a,b],[X262,a,b])?
[0] exit: suffix([a,b],[X178,X262,a,b])?
... in 3 reductions.

List = [X178,X262,a,b]
```

הרשימה [a,b] הינה סיומת של הרשימה List אשר אינה ידועה, וזאת גם משמעות התשובות שהתקבלו.

איור 9.14 – suffix([a,b], List)

```
tracing: ¦ ?- permutation(X,[a,b,c,d]).
[0] call: permutation(X80,[a,b,c,d])?
 [1] call: select(a,List180,Rest184)?
 [1] exit: select(a,[a¦Tail276],Tail276)?
 [2] call: permutation(Tail276,[b,c,d])?
  [3] call: select(b,Tail276,Rest405)?
  [3] exit: select(b,[b¦Tail497],Tail497)?
  [4] call: permutation(Tail497,[c,d])?
   [5] call: select(c,Tail497,Rest626)?
   [5] exit: select(c,[c¦Tail718],Tail718)?
   [6] call: permutation(Tail718,[d])?
    [7] call: select(d,Tail718,Rest847)?
    [7] exit: select(d,[d¦Tail939],Tail939)?
    [8] call: permutation(Tail939,[])?
    [8] exit: permutation([],[])?
   [6] exit: permutation([d],[d])?
  [4] exit: permutation([c,d],[c,d])?
 [2] exit: permutation([b,c,d],[b,c,d])?
[0] exit: permutation([a,b,c,d],[a,b,c,d])?
... in 9 reductions.

X = [a,b,c,d]
```

הרשימה [a,b,c,d] הינה תמורה של הרשימה X אשר אינה ידועה.
התשובה הראשונה שהתקבלה הינה התמורה עצמה. לאחר מכן, התכנית נכנסת ללולאה אינסופית. נמליץ לחזור לפתרון זה אחרי דיון במודל חישובי של תכנית לוגית שבפרק 10.

איור 9.15 – permutation(X, [a,b,c,d])

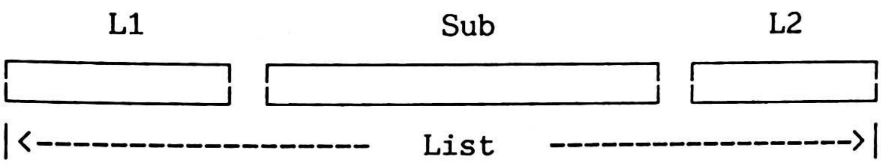
תרגיל 9.4

א. אתר את הפרוצדורה permutation באיור 9.20 של פרק זה ובאיור 22 של חטיבה רביעית, ועקוב אחרי מהלכה בשני המקרים.

ב. מצא הגדרה נוספת של היחס permutation, ללא select.

9.5 פירוק של רשימה באמצעות הפרוצדורה sublist - תרגול

בסעיף זה נמשיך את המטלה של הסעיף הקודם – הפעלת תכנית נתונה. התכנית של הסעיף הזה "מפרקת" רשימה לתת-רשימות. אומרים, כי Sub היא תת-רשימה (sublist) של רשימה List, אם הרשימה List משורשרת משלוש רשימות: Sub, L1 ו-L2 .



איור 9.16 – תת-רשימה

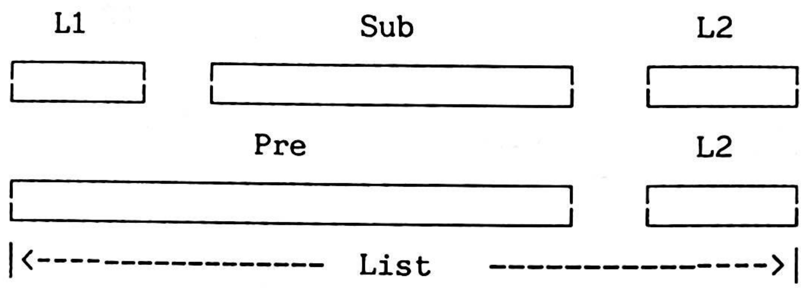
אין זו הגדרה פורמלית ולכן אין אנו מציינים אם Sub שונה מ-[] או מ-L. להלן נציג מספר אלטרנטיבות להגדרת תת-רשימה. הקורא מתבקש לבדוק אם אלטרנטיבות אלו שקולות, אם לאו. הפרוצדורות append, suffix, prefix, שבהן נעזרנו, בעת הגדרת הפרוצדורה sublist, הוגדרו בסעיף הקודם.

הסימונים של המשתנים שבפרוצדורות הם בהתאם לאיור 9.16. הרשימה Sub היא תת-רשימה של הרשימה List. הרשימה Pre היא שרשור הרשימות L1 ו-Sub, והיא מהווה קידומת של הרשימה List. בדומה, הרשימה Suff היא שרשור הרשימות Sub ו-L2, והיא סיומת של הרשימה List.

```
sublist1(Sub, List):- append(L1, Sub, Pre),
                      append(Pre, L2, List).

sublist2(Sub, List):- append(Pre, L2, List),
                      append(L1, Sub, Pre).
```

תכנית 9.6 – גירסאות 2,1 של הפרוצדורה sublist

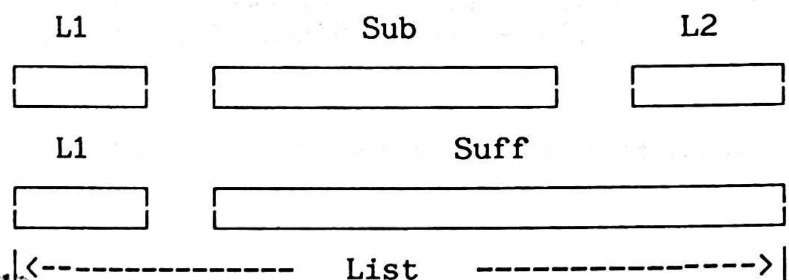


איור 9.17 – שרשור של L1 ו-Sub הוא קידומת של List

```
sublist3(Sub, List):- append(Sub, L2, Suff),
                      append(L1, Suff, List).

sublist4(Sub, List):- append(L1, Suff, List),
                      append(Sub, L2, Suff).
```

תכנית 9.6 (המשך) – גירסאות 4,3 של הפרוצדורה sublist

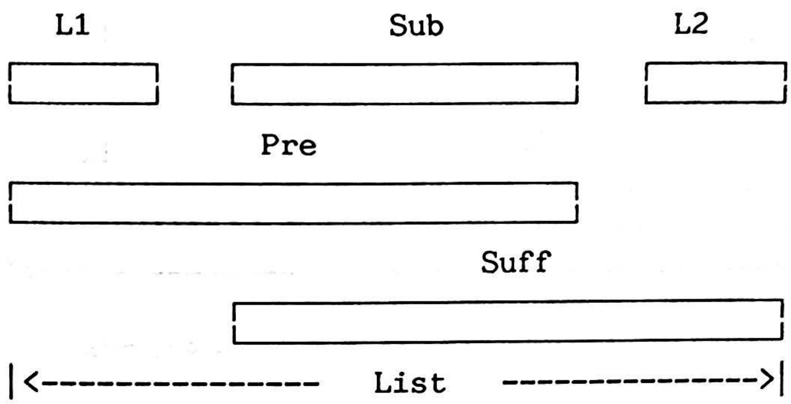


איור 9.18 – שרשור של Sub ו-L2 הוא סיומת של List

```
sublist5(Sub,List):-prefix(Sub,Suff),suffix(Suff,List).

sublist6(Sub,List):-suffix(Sub,Pre),prefix(Pre,List).
```

תכנית 9.6 (המשך) – גירסאות 6,5 של הפרוצדורה sublist



איור 9.19 – תת-רשימה Sub היא סיומת של Pre וקידומת של Suff

```
sublist7(Sub, List):- prefix(Pre, List), suffix(Sub, Pre).

sublist8(Sub, List):- prefix(Sub, List).
sublist8(Sub, [A|T]):- sublist8(Sub, T).

sublist9(Sub, List):- append(Sub, L2, List), Sub\=List.
sublist9(Sub, [A|T]):- sublist9(Sub, T), Sub\=[].
```

תכנית 9.6 (המשך) – גירסאות 9-7 של הפרוצדורה sublist

תרגיל 9.5

- ע[illegible]N בדוק מהן התשובות של פרולוג לכל אחת מהשאילתות האלה:

```
?- sublistN([b,c], [a,b,c,d]).
?- sublistN(X, [a,b,c,d]).
?- sublistN([a,b], List).
?- sublistN(X, List).
```

הערה: בתרגיל זה אתה מתבקש לענות על 36 (9★4) שאילתות, אשר לחלק מהן יש יותר מתשובה אחת. בעת הפעלת התכנית, אחרי הקשת ";", תתכן ההודעה out of stack. כמו כן, תתכן "נפילת התכנית". אל תתייאש, הטען את התכנית שוב והמשך בעבודתך. רשום את כל התשובות אשר קבלת. בפרק 10, לאחר שתכיר את המודל החישובי של תכנית לוגית, תתבקש לחזור לתרגיל זה.

9.6 עיצוב של תכנית לוגית: מיון רשימות

בסעיף זה נכתוב תכניות אחדות, אשר באמצעותן ימויינו איברי הרשימה. תכניות אלו מכונות בשם **תכניות מיון**.

מיון (sort) הוא פעולת סידור פריטים על פי קריטריון נתון. אם הפריטים הם מספרים, הקריטריון הזה יכול להיות הסדר העולה, או הסדר היורד שלהם. אם הפריטים הם אותיות, ניתן למיינם על פי הסדר האלפה-ביתי. באופן כללי ניתן לבטא את הקריטריון הזה באמצעות היחס precedes(A,B), שמשמעו: הפריט A קודם לפריט B. כדי לפשט את הדיון ולרכזו במיון בלבד, נקבל כי הרשימה מורכבת ממספרים, אשר יש למיין אותם לפי הסדר העולה:

precedes(A,B):-A=<B.

קיימים אלגוריתמים רבים למיון. לא נדון כאן בנושאי יעילותם. במרכז הדיון נעמיד את נושא העיצוב והכתיבה של תכנית לוגית. בשלב העיצוב מתארים את הצעדים הלוגיים לביצוע המשימה. השיטה הטבעית לעיצוב של תכנית בפרולוג היא **עיצוב מובנה מלמעלה למטה** (top-down structured design). נתחיל **מלמעלה**, דהיינו, נגדיר את המשימה ונתאר את צעדי הביצוע. בשלב הבא **נרד מטה**, דהיינו, נפרט ונעדן את הצעדים. בעת העידון נשמור על העקרון של **עיצוב מובנה** כלומר, נפרט את הצעדים כתת תכניות ופרוצדורות עצמאיות. את אשר אמרנו נייש בכתיבת תכניות מיון שלהלן.

9.6(א) מיון טבעי – Naive sort

המשימה היא למיין רשימת מספרים נתונה. את המשימה נבטא באמצעות היחס sort. את הרשימה הנתונה נסמן באמצעות המשתנה List, ואת הרשימה הממויינת – באמצעות המשתנה Sorted.
את המשימה נתאר כך: (sort(List, Sorted, משמע: צריך למיין את הרשימה List ולקבל את הרשימה Sorted.

את המשימה נבצע כך: נייצר את התמורות של הרשימה הנתונה, ונבדוק איזו מהן היא הרשימה הממויינת:

```
sort(List, Sorted):-
                        permutation(List, Sorted),
                        sorted(Sorted).
```

את היחס permutation הגדרנו בסעיף הקודם.
נפרט את הצעד הבא: (sorted(Sorted. רשימה בת פריט אחד היא רשימה ממויינת:

```
sorted([A]).
```

איך נבדוק, האם רשימה שמכילה שני פריטים או יותר, היא אכן ממויינת? ניעזר במבנה רקורסיבי של הרשימה: נבדוק אם שני פריטיה הראשונים מסודרים על פי הקריטריון הנבחר, ונמשיך בבדיקת הסדר של שאר הפריטים באמצעות חזרה רקורסיבית:

```
sorted([A, B|Rest]):-
                        precedes(A, B),
                        sorted([B|Rest]).
```

את היחס precedes נפרט בהתאם לנאמר בפתח הסעיף, כלומר, נחליפו ביחס "קטן או שווה".
להלן התכנית בשלמותה. בצד ימין הוספנו הסברים לצעדים שבוצעו.

כדי למיין רשימה List, יש
לבחור מבין תמורותיה את
הרשימה הממויינת
	`sort(List, Sorted):-`
	`    permutation(List, Sorted),`
	`    sorted(Sorted).`

ראה סעיף 9.3	`permutation(List, Sorted)`

תמורה בת איבר אחד ממויינת.
תמורה בת שניים ויותר איברים
ממויינת, אם שני איבריה הראשונים
מסודרים, וזנבה ממויין.
	`sorted([A]).`
	`sorted([A, B|Rest]):-`
	`    A=<B,`
	`    sorted([B|Rest]).`

תכנית 9.7 – Naive sort

את המשימה ביצענו בדרך של "ייצר ובדוק" (generate and test). ייצרנו את התמורה ובדקנו אם היא הפתרון. אין זו שיטה יעילה למיון, שכן ייצור התמורות איננו קשור למשימה, ואפשר שנייצר תמורות רבות עד שנגיע לתמורה הנדרשת. בסעיף הבא יוצג trace של כל אחת מתכניות המיון של הסעיף הזה והקורא מוזמן להיווכח, כי המיון באמצעות התכנית Naive sort אינו היעיל ביותר. הסיבה שבגללה החלטנו לדון בתכנית הזאת, היא צורתה הפשוטה. תיארנו את המשימה ומייד כתבנו תכנית ברורה ותמציתית. נעיז ונאמר: אין זו בעייתנו שהמחשב של הדור הנוכחי מייצר תמורות באופן סדרתי בזו אחר זו. האם נחכה עד לבואו של המחשב המקבילי של הדור הבא? לא ולא. בפרולוג ניתן לכתוב תכניות קצרות ויעילות כאחד. נעבור לתכנית המיון הבאה.

9.6(ב) מיון ע"י שיבוץ – Insertion sort

המיון ע"י שיבוץ "מנצל" את המבנה הרקורסיבי של הרשימה: נניח שזנב הרשימה כבר ממויין; נשאר לשבץ את האיבר הראשון במקום הנכון בהתאם לסדר.
נתאר את הצעדים למיון רשימה לא-ריקה בשפה יותר פורמלית:

א. נמיין את זנבה,
ב. נשבץ את ראשה אל תוך זנבה הממויין כך, שהרשימה שתתקבל תהיה הרשימה הממויינת.

מכאן נעבור לכתיבת התכנית:

```
isort([A|Tail], Sorted):-
                 isort(Tail, SortedTail),
                 insert(A, SortedTail, Sorted).
```

הרשימה הריקה היא אכן רשימה ממויינת:

```
isort([], []).
```

את השיבוץ נבצע כך: נשווה בין המספר המשובץ לבין ראשו של הזנב הממויין. אם המספר המשובץ איננו גדול מראשו של הזנב, נעמיד אותו בראש הרשימה הממויינת:

```
insert(A, [H|Rest], [A, H|Rest]):-
                                  A=<H.
```

אחרת – נשבץ אותו במקום הנכון בתוך הרשימה Rest. הכיצד? אין צורך לפרט. החוק הרקורסיבי יבצע את השיבוץ ללא עזרתנו:

```
insert(A, [H|Rest], [H|New):-
                            A>H,
                            insert(A, Rest, New).
```

איך בכל זאת התבצע השיבוץ של A בתוך Rest? החוק הרקורסיבי ישווה בין A לבין ראשו של Rest. במידת הצורך, ראשו של Rest "יקוצץ" ותעשה השוואה בין A לבין ראש השארית של Rest, וכן הלאה. התנאי הגבולי, אם כן, הוא:

```
insert(A, [], [A]).
```

להלן התכנית בשלמותה:

```
isort([A|Tail], Sorted):-
                 isort(Tail, SortedTail),
                 insert(A, SortedTail, Sorted).
isort([], []).

insert(A, [H|Rest], [A,H|Rest]):-
                 A=<H.
insert(A, [H|Rest], [H|New]):-
                 A>H,
                 insert(A, Rest, New).
insert(A, [], [A]).
```

תכנית 9.8 – Insertion sort

בתכנית הזאת ביצענו את המשימה בדרך זו: היחס isort מפריד בין איברי הרשימה, והיחס insert בונה את הרשימה הממויינת.

9.6(ג) מיון בועות – Bubble sort

במיון בועות משווים בין האיבר הראשון של הרשימה לבין האיבר השני, ואם סדרם הפוך – מחליפים ביניהם. אחר כך משווים בין האיבר השני (החדש) לבין השלישי, וכן הלאה. בצורה זאת עולה האיבר הגדול ביותר וצף, בדומה לבועת אוויר במים, עד שהוא מגיע לסוף הרשימה. לאחר מכן פעולת ההשוואה חוזרת על עצמה.

את מיון הבועות נבטא באמצעות היחס busort(List,Sorted), ואת השוואתם והחלפתם של שני מספרים עוקבים נבטא באמצעות היחס change(List,New). את משימת המיון נתאר באמצעות החוק הבא:

```
busort(List, Sorted):-
          change(List, New),
          busort(New, Sorted).
```

היחס change(List,New) מתאר את החיפוש אחר שני מספרים עוקבים, אשר סדרם הפוך:

```
change([A, B|Rest], [B, A|Rest]):-
          A>B.
change([A|Rest], [A|NewRest]):-
          change(Rest, NewRest).
```

אם ברשימה List אין זוגות מספרים עוקבים אשר סדרם הפוך, תהיה הרשימה List ממויינת:

```
busort(List, List):- sorted(List).
```

להלן התכנית בשלמותה:

```
busort(List, Sorted):-
          change(List, New),
          busort(New, Sorted).
busort(List, List):- sorted(List).

change([A, B|Rest], [B, A|Rest]):-
          A>B.
change([A|Rest], [A|NewRest]):-
          change(Rest, NewRest).

sorted(List)        -       ראה תכנית 9.7
```

תכנית 9.9 – Bubble sort

היחס change ממיין את הרשימה, והיחס busort מכוון את פעולתו.

הנה תכנית נוספת, אשר בה פעולת המיון מבוטאת באמצעות היחס, המוכר לנו מהסעיף הקודם:

```
busort1(List, Sorted):-
            append(Pre, [X,Y|Rest], List),
            X>Y,
            append(Pre, [Y,X|Rest], NewList),
            busort1(NewList, Sorted).
busort1(List, List):- sorted(List).

append          -          ראה פרוצדורה 9.3(א)
sorted          -          ראה תכנית 9.7
```

תכנית 9.10 – Bubble sort using append

9.6(ד) מיון מהיר – Quicksort

מיון מהיר נחשב למיון יעיל ומתוחכם. המפתח של האלגוריתם למיון מהיר זכה בפרס טיורינג (ACM Turing Award). להלן נוהל המיון:

א. נוציא איבר כלשהו מן הרשימה.
ב. את הרשימה נפרק לשתי רשימות:
– האחת מכילה איברים הקטנים מזה שהוצא.
– השניה מכילה איברים הגדולים מזה שהוצא.
ג. נמיין כל אחת מהרשימות.
ד. נשרשר אותן, כאשר האיבר שהוצא נמצא בין הרשימות. כלומר, במקומו הנכון.

בדרך כלל מוציאים מן הרשימה את האיבר הראשון.
ניישם את שיטת המיון המהיר באמצעות התכנית הבאה:

```
quicksort([], []).
quicksort([A|Tail], Sorted):-
        split(A, Tail, Smaller, Bigger),
        quicksort(Smaller, S),
        quicksort(Bigger, B),
        append(S, [A|B], Sorted).

split(A, [], [], []).
split(A, [B|T], [B|Small], Big):-
        A>B,
        split(A, T, Small, Big).
split(A, [B|T], Small, [B|Big]):-
        A=<B,
        split(A, T, Small, Big).

append - 9.3 ראה סעיף
```

תכנית 9.11 – Quicksort

הסבר:

הפרוצדורה split מפרקת את זנב הרשימה Tail לשתי רשימות: Smaller ו-Bigger. הרשימה Smaller מכילה את האיברים שאינם גדולים מ-A, והרשימה Bigger מכילה את אלה הגדולים מ-A. החוק הראשון של הפרוצדורה split קובע, כי פירוק רשימה ריקה נותן שתי רשימות ריקות. החוק השני של split מגדיר רקורסיבית את הפירוק, כאשר A גדול מהראש B של Tail. החוק השלישי מופעל, כאשר A אינו גדול מ-B.

9.7 תכניות מיון – הריצה והמעקב (trace)

תכניות המיון, אשר הצגנו לפניך, מכילות 11 פרוצדורות, חלקן רקורסיביות. בפרוצדורות אלו תוכל להיעזר כשתכתוב תכניות נוספות. הרצנו את תכניות המיון בגירסת WISDOM PROLOG ועקבנו אחרי מהלכן. להלן תוצאות הריצה והמעקב של חלק מהתכניות. אתה מתבקש לעיין בהן היטב ולהסבירן.

ראה איור 9.20 בעמוד הבא

```
tracing: ¦ ?- isort([2,1,3],Sorted).
[0] call: isort([2,1,3],Sorted87)?w
 [1] call: isort([1,3],SortedTail190).
  [2] call: isort([3],SortedTail279).
   [3] call: isort([],SortedTail368).
   [3] exit: isort([],[]).
   [4] call: insert(3,[],Sorted367).
   [4] exit: insert(3,[],[3]).
  [2] exit: isort([3],[3]).
  [5] call: insert(1,[3],Sorted278).
   [6] call: 1 =< 3.
   [6] exit: 1 =< 3.
  [5] exit: insert(1,[3],[1,3]).
 [1] exit: isort([1,3],[1,3]).
 [6] call: insert(2,[1,3],Sorted189).
  [7] call: 2 =< 1.
  [7] fail: 2 =< 1.
  [7] call: 2 > 1.
  [7] exit: 2 > 1.
  [7] call: insert(2,[3],New984).
   [8] call: 2 =< 3.
   [8] exit: 2 =< 3.
  [7] exit: insert(2,[3],[2,3]).
 [6] exit: insert(2,[1,3],[1,2,3]).
[0] exit: isort([2,1,3],[1,2,3]).
 .. in 8 reductions.

Sorted = [1,2,3]
```

איור 9.21 – מיון של הרשימה [2,1,3] על-ידי שיבוץ (תכנית 9.8)

```
tracing: ¦ ?- quicksort([2,1,3],Sorted).
[0] call: quicksort([2,1,3],Sorted87)?u
 [1] call: split(2,[1,3],Smaller190,Bigger191).
  [2] call: 2 > 1.
  [2] exit: 2 > 1.
  [2] call: split(2,[3],Small288,Big289).
   [3] call: 2 > 3.
   [3] fail: 2 > 3.
   [3] call: 2 =< 3.
   [3] exit: 2 =< 3.
   [3] call: split(2,[],Small478,Big480).
   [3] exit: split(2,[],[],[]).
  [2] exit: split(2,[3],[],[3]).
 [1] exit: split(2,[1,3],[1],[3]).
 [4] call: quicksort([1],S192).
  [5] call: split(1,[],Smaller868,Bigger869).
  [5] exit: split(1,[],[],[]).
  [6] call: quicksort([],S870).
  [6] exit: quicksort([],[]).
  [7] call: quicksort([],B871).
  [7] exit: quicksort([],[]).
  [8] call: aappend([],[1],Sorted867).
  [8] exit: aappend([],[1],[1]).
 [4] exit: quicksort([1],[1]).
 [9] call: quicksort([3],B193).
  [10] call: split(3,[],Smaller1489,Bigger1490).
  [10] exit: split(3,[],[],[]).
  [11] call: quicksort([],S1491).
  [11] exit: quicksort([],[]).
  [12] call: quicksort([],B1492).
  [12] exit: quicksort([],[]).
  [13] call: aappend([],[3],Sorted1488).
  [13] exit: aappend([],[3],[3]).
 [9] exit: quicksort([3],[3]).
 [14] call: aappend([1],[2,3],Sorted189).
  [15] call: aappend([],[2,3],L32104).
  [15] exit: aappend([],[2,3],[2,3]).
 [14] exit: aappend([1],[2,3],[1,2,3]).
[0] exit: quicksort([2,1,3],[1,2,3]).
... in 16 reductions.

Sorted = [1,2,3]
```

איור 9.22 – מיון "מהיר" של הרשימה [2,1,3] (תכנית 9.11)

```
tracing: ¦ ?- sort([2,1,3],Sorted).
[0] call: sort([2,1,3],Sorted87)?
 [1] call: permutation([2,1,3],Sorted187).
  [2] call: select(X274,[2,1,3],Rest276).
  [2] exit: select(2,[2,1,3],[1,3]).
  [3] call: permutation([1,3],Permut275).
   [4] call: select(X480,[1,3],Rest482).
   [4] exit: select(1,[1,3],[3]).
   [5] call: permutation([3],Permut481).
    [6] call: select(X686,[3],Rest688).
    [6] exit: select(3,[3],[]).
    [7] call: permutation([],Permut687).
    [7] exit: permutation([],[]).
   [5] exit: permutation([3],[3]).
  [3] exit: permutation([1,3],[1,3]).
 [1] exit: permutation([2,1,3],[2,1,3]).
 [8] call: sorted([2,1,3]).
  [9] call: 2 =< 1.
  [9] fail: 2 =< 1.
 [8] fail: sorted([2,1,3]).
 [1] redo: permutation([2,1,3],[2,1,3]).
  [3] redo: permutation([1,3],[1,3]).
   [5] redo: permutation([3],[3]).
    [7] redo: permutation([],[]).
     [8] call: select(X892,[],Rest894).
     [8] fail: select(X892,[],_894).
    [7] fail: permutation([],_687).
    [6] redo: select(3,[3],[]).
     [7] call: select(X773,[],T778).
     [7] fail: select(X773,[],_778).
    [6] fail: select(_686,[3],_688).
   [5] fail: permutation([3],_481).
   [4] redo: select(1,[1,3],[3]).
    [5] call: select(X567,[3],T572).
    [5] exit: select(3,[3],[]).
   [4] exit: select(3,[1,3],[1]).
   [6] call: permutation([1],_481).
    [7] call: select(X804,[1],Rest806).
    [7] exit: select(1,[1],[]).
    [8] call: permutation([],Permut805).
    [8] exit: permutation([],[]).
   [6] exit: permutation([1],[1]).
  [3] exit: permutation([1,3],[3,1]).
 [1] exit: permutation([2,1,3],[2,3,1]).
 [9] call: sorted([2,3,1]).
  [10] call: 2 =< 3.
  [10] exit: 2 =< 3.
  [10] call: sorted([3,1]).
   [11] call: 3 =< 1.
   [11] fail: 3 =< 1.
  [10] fail: sorted([3,1]).
  [10] redo: 2 =< 3.
  [10] fail: 2 =< 3.
 [9] fail: sorted([2,3,1]).
```

(המשך)

```
 [1] redo: permutation([2,1,3],[2,3,1]).
  [3] redo: permutation([1,3],[3,1]).
   [6] redo: permutation([1],[1]).
    [8] redo: permutation([],[]).
     [9] call: select(X1010,[],Rest1012).
     [9] fail: select(X1010,[],_1012).
    [8] fail: permutation([],_805).
    [7] redo: select(1,[1],[]).
     [8] call: select(X891,[],T896).
     [8] fail: select(X891,[],_896).
    [7] fail: select(_804,[1],_806).
   [6] fail: permutation([1],_481).
   [4] redo: select(3,[1,3],[1]).
    [5] redo: select(3,[3],[]).
     [6] call: select(X648,[],T653).
     [6] fail: select(X648,[],_653).
    [5] fail: select(_567,[3],_572).
   [4] fail: select(_480,[1,3],_482).
  [3] fail: permutation([1,3],_275).
  [2] redo: select(2,[2,1,3],[1,3]).
   [3] call: select(X361,[1,3],T366).
   [3] exit: select(1,[1,3],[3]).
  [2] exit: select(1,[2,1,3],[2,3]).
  [4] call: permutation([2,3],_275).
   [5] call: select(X598,[2,3],Rest600).
   [5] exit: select(2,[2,3],[3]).
   [6] call: permutation([3],Permut599).
    [7] call: select(X804,[3],Rest806).
    [7] exit: select(3,[3],[]).
    [8] call: permutation([],Permut805).
    [8] exit: permutation([],[]).
   [6] exit: permutation([3],[3]).
  [4] exit: permutation([2,3],[2,3]).
 [1] exit: permutation([2,1,3],[1,2,3]).
 [9] call: sorted([1,2,3]).
  [10] call: 1 =< 2.
  [10] exit: 1 =< 2.
  [10] call: sorted([2,3]).
   [11] call: 2 =< 3.
   [11] exit: 2 =< 3.
   [11] call: sorted([3]).
   [11] exit: sorted([3]).
  [10] exit: sorted([2,3]).
 [9] exit: sorted([1,2,3]).
[0] exit: sort([2,1,3],[1,2,3]).
... in 12 reductions.

Sorted = [1,2,3]
```

בתכנית זו נוצרו התמורות האלו:

[2,1,3]
[2,3,1]
[1,2,3]

התמורה האחרונה הינה הרשימה הממויינת.

איור 9.20 – מיון טבעי של הרשימה [2,1,3] (תכנית 9.7)

```
tracing: ¦ ?- busort([3,2,1],Sorted).
[0] call: busort([3,2,1],Sorted87)?
 [1] call: change([3,2,1],New188).
  [2] call: 3 > 2.
  [2] exit: 3 > 2.
 [1] exit: change([3,2,1],[2,3,1]).
 [2] call: busort([2,3,1],Sorted187).
  [3] call: change([2,3,1],New494).
   [4] call: 2 > 3.
   [4] fail: 2 > 3.
   [4] call: change([3,1],NewRest583).
    [5] call: 3 > 1.
    [5] exit: 3 > 1.
   [4] exit: change([3,1],[1,3]).
  [3] exit: change([2,3,1],[2,1,3]).
  [5] call: busort([2,1,3],Sorted493).
   [6] call: change([2,1,3],New917).
    [7] call: 2 > 1.
    [7] exit: 2 > 1.
   [6] exit: change([2,1,3],[1,2,3]).
   [7] call: busort([1,2,3],Sorted916).
    [8] call: change([1,2,3],New1223).
     [9] call: 1 > 2.
     [9] fail: 1 > 2.
     [9] call: change([2,3],NewRest1312).
      [10] call: 2 > 3.
      [10] fail: 2 > 3.
      [10] call: change([3],NewRest1392).
       [11] call: change([],NewRest1472).
       [11] fail: change([],_1472).
      [10] fail: change([3],_1392).
     [9] fail: change([2,3],_1312).
    [8] fail: change([1,2,3],_1223).
    [8] call: sorted([1,2,3]).
     [9] call: 1 =< 2.
     [9] exit: 1 =< 2.
     [9] call: sorted([2,3]).
      [10] call: 2 =< 3.
      [10] exit: 2 =< 3.
      [10] call: sorted([3]).
      [10] exit: sorted([3]).
     [9] exit: sorted([2,3]).
    [8] exit: sorted([1,2,3]).
   [7] exit: busort([1,2,3],[1,2,3]).
  [5] exit: busort([2,1,3],[1,2,3]).
 [2] exit: busort([2,3,1],[1,2,3]).
[0] exit: busort([3,2,1],[1,2,3]).
... in 11 reductions.

Sorted = [1,2,3]
```

איור 9.23 – מיון של [3,2,1]
באמצעות "בועות" (תכנית 9.9)

```
tracing: ¦ ?- busort2([3,2,1],Sorted).
[0] call: busort2([3,2,1],Sorted87)?
 [1] call: aappend(Pre188,[X190,Y192¦Rest193],[3,2,1]).
 [1] exit: aappend([],[3,2,1],[3,2,1]).
 [2] call: 3 > 2.
 [2] exit: 3 > 2.
 [2] call: aappend([],[2,3,1],NewList196).
 [2] exit: aappend([],[2,3,1],[2,3,1]).
 [3] call: busort2([2,3,1],Sorted187).
  [4] call: aappend(Pre633,[X635,Y637¦Rest638],[2,3,1]).
  [4] exit: aappend([],[2,3,1],[2,3,1]).
  [5] call: 2 > 3.
  [5] fail: 2 > 3.
  [4] redo: aappend([],[2,3,1],[2,3,1]).
   [5] call: aappend(L1732,[_635,_637¦_638],[3,1]).
   [5] exit: aappend([],[3,1],[3,1]).
  [4] exit: aappend([2],[3,1],[2,3,1]).
  [6] call: 3 > 1.
  [6] exit: 3 > 1.
  [6] call: aappend([2],[1,3],NewList641).
   [7] call: aappend([],[1,3],L31083).
   [7] exit: aappend([],[1,3],[1,3]).
  [6] exit: aappend([2],[1,3],[2,1,3]).
  [8] call: busort2([2,1,3],Sorted632).
   [9] call: aappend(Pre1314,[X1316,Y1318¦Rest1319],[2,1,3]).
   [9] exit: aappend([],[2,1,3],[2,1,3]).
   [10] call: 2 > 1.
   [10] exit: 2 > 1.
   [10] call: aappend([],[1,2,3],NewList1322).
   [10] exit: aappend([],[1,2,3],[1,2,3]).
   [11] call: busort2([1,2,3],Sorted1313).
    [12] call: aappend(Pre1759,[X1761,Y1763¦Rest1764],[1,2,3]).
    [12] exit: aappend([],[1,2,3],[1,2,3]).
    [13] call: 1 > 2.
    [13] fail: 1 > 2.
    [12] redo: aappend([],[1,2,3],[1,2,3]).
     [13] call: aappend(L11858,[_1761,_1763¦_1764],[2,3]).
     [13] exit: aappend([],[2,3],[2,3]).
    [12] exit: aappend([1],[2,3],[1,2,3]).
    [14] call: 2 > 3.
    [14] fail: 2 > 3.
    [12] redo: aappend([1],[2,3],[1,2,3]).
     [13] redo: aappend([],[2,3],[2,3]).
      [14] call: aappend(L11939,[_1761,_1763¦_1764],[3]).
       [15] call: aappend(L12020,[_1761,_1763¦_1764],[]).
       [15] fail: aappend(L12020,[_1761,_1763¦_1764],[]).
      [14] fail: aappend(_1939,[_1761,_1763¦_1764],[3]).
     [13] fail: aappend(_1858,[_1761,_1763¦_1764],[2,3]).
    [12] fail: aappend(_1759,[_1761,_1763¦_1764],[1,2,3]).
   [11] exit: busort2([1,2,3],[1,2,3]).
  [8] exit: busort2([2,1,3],[1,2,3]).
 [3] exit: busort2([2,3,1],[1,2,3]).
[0] exit: busort2([3,2,1],[1,2,3]).
... in 12 reductions.

Sorted = [1,2,3]
```

התנאי הגבולי של תכנית 9.10
הוחלף: busort2(Sorted, Sorted).

איור 9.24 – "בועות" של [3,2,1] בעזרת append

פרק 10

מנגנון ההוכחה של פרולוג

בסעיפים 7.3, 7.4, 7.5 תיארנו תהליך של הסקת מסקנות על-ידי מערכת הפרולוג מנקודת ראות לוגית. בפרק זה נתאר את התהליך הזה במונחים של ביצוע. לשם כך נדון במודל חישובי של תכנית לוגית ונכיר את יישומו בפרולוג. בסעיפים 10.1 ו-10.2 נכיר את המושגים **הצבה** ו**מאורע**. המודל החישובי של תכנית לוגית מבוסס על אלגוריתם של איחוד. במושגים **התאמה** ו**איחוד** נדון בסעיפים 10.3 ו-10.4. את המודל נכיר בסעיף 10.5, ואת יישומו על ידי מערכת הפרולוג – בסעיפים 10.6 ו-10.7.

10.1 הצבה ומאורע

ההצבה היא חלק של תהליך חישוב השאילתה. לפני שנדון במושג ההצבה, נחזור ונעיין במרכיבי התכנית הלוגית. המרכיבים – עובדה; חוק ושאילתה, הם נוסחאות של לוגיקת היחסים. ליתר דיוק, הם נוסחאות של לוגיקת היחסים מסדר ראשון. נציין כי בלוגיקה מסדר ראשון הפרדיקט הוא אטום ולא משתנה, משמע – הביטויים מהסוג X(tom,liz) אינם חוקיים. הנוסחאות האלו בנויות בצורה תחבירית אחידה: לחוק יש ראש וגוף, לעובדה יש ראש ואין גוף, לשאילתה יש גוף ואין ראש. את העובדה ניתן לפרש כחוק, אשר גופו מכיל פרדיקט אחד בלבד: true.

פיסקה	ראש	אם	גוף
עובדה	א	-:	
עובדה	א	-:	אמת
חוק	א	-:	ב,ג,ד
שאילתה		-:	ב,ג,ד

Clause	Head	IF	Body
Fact	a	:-	
Fact	a	:-	true
Rule	a	:-	b,c,d
Query		:-	b,c,d

איור 10.1 – מרכיבים של תכנית לוגית

המשתנים של תכנית לוגית הם משתנים מכומתים ועל כן נוסחאות של תכנית לוגית הן פסוקים. משתני השאילתה מכומתים קיומי, משתני העובדה והחוק מכומתים כללי. את הכמתים אין כותבים באופן מפורש. הפסוקים של לוגיקת היחסים, אשר הכמתים אינם מופיעים בהם בצורה מפורשת, מכונים בשם **פיסקאות** (clauses). באמצעות תכונות השקילות, שאותן למדנו בפרקים 2 ו-6, ניתן להפוך כל פסוק לפיסקה, השקולה לפסוק הזה מבחינה סמנטית. את תהליך ההפיכה של פסוק לפיסקה ניתן למצוא בספרי לימוד רבים, למשל, ב-[Winston, 1984] ולא נדון בכך כאן. את הפיסקאות מהסוג "ראש אם גוף", אשר ראשן ריק או כולל נוסחה אטומית אחת בלבד, מכנים **פיסקאות של הורן** (Horn clauses). מכאן, שהמרכיבים של תכנית לוגית הם פיסקאות של הורן.

פיסקה של תכנית לוגית, אשר אינה כוללת משתנים, נקראת **פיסקה בסיסית**. בדומה, ביטוי ללא משתנים הוא **ביטוי בסיסי**, עובדה ללא משתנים היא **עובדה בסיסית** ושאילתה ללא משתנים היא **שאילתה בסיסית**.

ערך-אמת של עובדה ושל חוק הוא תמיד T (אמת). את ערך האמת של השאילתה יש לחשב, וכחלק מתהליך החישוב יש לבצע הצבות. ההצבה היא קבוצה סופית של משוואות שצורתן היא: X = term.

להלן ההגדרה הפורמלית:

הגדרה 10.1

ההצבה (substitution) θ היא קבוצה של משוואות:

```
{X1=term1,  X2=term2,  ...,  XN = termN}
```

אשר בה:

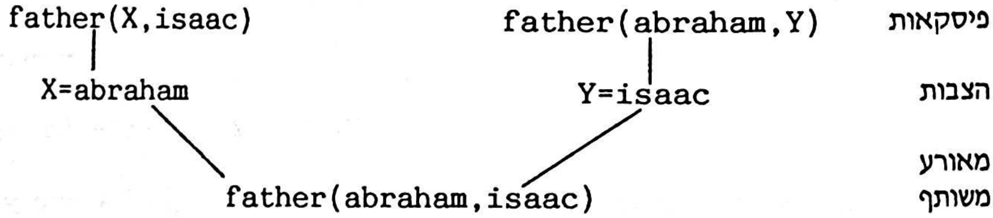
- X1, X2,...,XN הם משתנים שונים, (N≥1).
- term1, term2,...,termN הם ביטויים (לאו דווקא שונים ולאו דווקא בסיסיים).

הערה: בפרולוג העברי כותבים את המשוואות כך: X1=ביטוי1, X1=ביטוי2, וכן הלאה, מימין לשמאל.

הגדרה 10.2

תהי a פיסקה ו-θ – הצבה. את תוצאת ההצבה θ בתוך a מסמנים ב-$a\theta$ ומכנים **מאורע** (instance), **החלפה**, או **מקרה** של a. להבא נשתמש במונח **מאורע**.

המונח הבא שנגדיר יהיה **מאורע משותף**. האיור 10.2 מתאר את המונח הזה באופן בלתי פורמלי:



איור 10.2 – מאורע משותף

הגדרה 10.3

תהיינה a ו-b פיסקאות. נאמר, כי הפיסקה c היא **מאורע משותף** (common instance) של a ו-b, אם c היא מאורע של a ומאורע של b, כלומר, אם קיימות שתי הצבות, θ_1 ו-θ_2, כך, ש-$c=a\theta_1$ ו- $c=b\theta_2$.

במילים אחרות, ל-a ול-b יש מאורע משותף אם מאורע $a\theta_1$ של a זהה תחבירית למאורע $b\theta_2$ של b. להלן דוגמאות של מאורע משותף.

דוגמה 10.1

נתונה הפיסקה a:

```
ancestor(abraham, Y):- father(abraham, Link),
                       ancestor(Link, Y).
```

נתונה הפיסקה b:

```
ancestor(X, kobi):- father(X, Link),
                    ancestor(Link, kobi)
```

נבצע את ההצבות שלהלן:

	ביטוי		משתנה
הצבה θ_1 לתוך הפיסקה a:	kobi	=	Y
הצבה θ_2 לתוך הפיסקה b:	abraham	=	X

המאורעות $a\theta_1$ ו-$b\theta_2$ זהים תחבירית, משמע שלפיסקה a ו-b יש מאורע משותף $a\theta_1=b\theta_2$ והוא:

```
ancestor(abraham, kobi):- father(abraham, Link),
                          ancestor(Link, kobi).
```

דוגמה 10.2

הפיסקאות של הדוגמה הזאת הן של הפרולוג העברי, ולכן יש לקרוא את ההצבות מימין לשמאל.

נתונה הפיסקה a:

משפחה(X, אם(צבר (Y), Z),

נתונה הפיסקה b:

משפחה(אב(A, B), W , U).

לשתי פיסקאות אלו יש מאורע משותף c, אשר מתקבל כתוצאת ביצוע של ההצבות שלהלן:

	משתנה		ביטוי
ההצבה θ1 בתוך פיסקה a:	X	=	אב(A, B)
	Z	=	Z1
ההצבה θ2 בתוך פיסקה b:	W	=	אם(צבר, Y)
	U	=	Z1

המאורע המשותף c הוא:

משפחה(אב(A, B), אם(צבר, Y), Z1)

לפיסקאות a ו-b ישנם מאורעות משותפים נוספים. נעיין בפיסקה אשר נסמנה ב-d:

משפחה(אב(צבר, יורם), אם(צבר, רינה), ילדים([])).

הפיסקה d מתקבלת מ-a באמצעות ההצבה θ3:

X = אב(צבר, יורם)
Y = רינה
Z = ילדים([]).

הפיסקה d מתקבלת מ-b באמצעות ההצבה θ4:

A = צבר
B = יורם
W = אם(צבר, רינה)
U = ילדים([]).

10.2 המאורע המשותף הכללי ביותר

נסמן ב-a את הפיסקה: father(X,Y), ונבצע בה את ההצבות האלה:

	ביטוי		משתנה
ההצבה θ1 (בתעתיק אנגלי):	X1	=	X
	Y1	=	Y

	ביטוי		משתנה
ההצבה θ2 (בתעתיק אנגלי) .	abraham	=	X
	Y1	=	Y

לאחר ההצבה קיבלנו שני מאורעות:

המאורע aθ1:	father(X1, Y1)
המאורע bθ2:	father(abraham, Y1)

בין שני המאורעות האלו יש הבדל מהותי. לאחר שנבצע בתוך aθ1 את ההצבה שבהמשך, נחזור אל הפיסקה father(X,Y). יוצא אם כן כי הפיסקה aθ1 היא המאורע של a, והפיסקה a היא המאורע של aθ1. משמע, שאת הפיסקה aθ1 קיבלנו כתוצאה של החלפת השמות של משתני הפיסקה a.

ההצבה:	ביטוי		משתנה
	X	=	X1
	Y	=	Y1

לעומת זאת, לא ניתן לחזור מן הפיסקה aθ2, אשר היא father(abraham, Y1). אל הפיסקה a, אשר היא father(X,Y). על פי הגדרת ההצבה ניתן להציב את הביטוי abraham תחת משתנה X אך לא להיפך.

הגדרה 10.4

תהיינה a ו-b שתי פיסקאות.
אומרים, כי b היא **הצורה האלפביתית** (alphabetic variant) של a, אם b היא המאורע של a ו-a היא מאורע של b.
אומרים, כי a היא **יותר כללית** (more general) מ-b, אם b היא מאורע של a, אבל a איננה מאורע של b.

דוגמה 10.3

הפיסקה father(X1, Y1) היא **הצורה האלפביתית** של father(X, Y).
הפיסקה father(X, Y) **יותר כללית** מהפיסקה father(abraham, isaac).

הגדרה 10.5

המאורע aθ של הפיסקה a מכונה **מאורע בסיסי** (ground instance), אם aθ היא פיסקה בסיסית (כלומר ב-aθ אין משתנים).

דוגמה 10.4

הפיסקה father(abraham, isaac) היא **המאורע הבסיסי** של father(abraham, Y1) ושל father(X, Y). המאורע הבסיסי father(abraham, isaac) הוא **סופי** במובן שאין לו מאורעות.

נחזור אל דוגמה 10.2, בה מצאנו שני מאורעות משותפים c ו-d עבור זוג הפיסקאות a ו-b. נציין שהפיסקה c **יותר כללית** מן הפיסקה d. ניתן למצוא מאורעות נוספים, המשותפים ל-a ול-b, כדוגמת המאורע e שלהלן:

המאורע המשותף e:

משפחה(אב(צבר, B), אם(צבר, Y), 1Z)

מאורע e התקבל מ-a באמצעות ההצבה θ5:

משתנה		ביטוי
X	=	אב(צבר, B)
Z	=	1Z

מאורע e התקבל מ-b באמצעות ההצבה θ6:

משתנה		ביטוי
A	=	צבר
W	=	אם(צבר, Y)
U	=	1Z

נציין, כי הפיסקה c **יותר כללית** מן הפיסקה e, והפיסקה e **יותר כללית** מן הפיסקה d. הפיסקה d היא **המאורע הבסיסי** של כל אחת מהפיסקאות: e,c,b,a.

הגדרה 10.6

תהיינה a ו-b שתי פיסקאות. המאורע המשותף c של a ושל b נקרא **הכללי ביותר**, אם הוא יותר כללי מכל מאורע אחר, המשותף ל-a ול-b.

נציין בהקשר לכך:

א. אם לשתי פיסקאות a ו-b יש מאורע משותף אחד לפחות, אז יש להן המאורע המשותף הכללי ביותר.

ב. המאורע המשותף הכללי ביותר הוא אחד ויחיד עד דיוק של צורה אלפביתית. כלומר, אם לפיסקאות a ו-b יש שני מאורעות משותפים כלליים ביותר, אז אחד מהמאורעות הוא הצורה האלפביתית של המאורע האחר.

10.3 התאמה

נעיין בשתי הפיסקאות a ו-b של דוגמה 10.2 בסעיף הקודם. את המאורע המשותף הכללי ביותר c מצאנו בדרך זו:

- את המשוואה X=אב(B,A) מצאנו על ידי ההשוואה בין הארגומנט הראשון של הפיסקה a לבין הארגומנט הראשון של הפיסקה b.

- את המשוואה W=אם(צבר,Y) מצאנו אחרי שהשווינו את הארגומנט השני בשתי הפיסקאות, וכך הלאה.

הגדרה 10.7

בהינתן שני ביטויים Term1 ו-Term2 אנו אומרים, כי יש ביניהם **התאמה** (matching), אם:

- Term1 זהה מבחינה תחבירית ל-Term2,
  או:
- קיימות שתי הצבות, האחת לתוך Term1 והאחרת לתוך Term2, כך שאם נבצע אותן יהיו הביטויים זהים מבחינה תחבירית.

בסעיף הקודם הגדרנו את המאורע המשותף של שתי פיסקאות (ראה את ההגדרה 10.3). אם נגדיר בצורה דומה את המאורע המשותף של שני ביטויים ניתן יהיה לנסח את ההגדרה כך:

הגדרה 10.7(א)

בין שני ביטויים יש **התאמה** אם הם זהים מבחינה תחבירית או אם יש להם מאורע משותף.

בדוגמה 10.2 מצאנו אם כן, כי יש התאמה בין הביטויים האלה:

```
בין   X               לבין   אב(A, B)
בין   אם(צבר, Y)      לבין   W.
בין   Z               לבין   U.
```

כמו כן מצאנו, כי קיימת התאמה בין

משפחה(X, אם(צבר, Y), Z)

לבין

משפחה(אב(A, B), W, U).

להלן דוגמאות נוספות:

דוגמה 10.5

בפרק 7 של החטיבה השניה חישבנו את התשובה לשאילתה סופר(X) וראינו, כי יש התאמה בין הביטויים האלה:

```
בין   לומד(X, ספרות)   לבין   לומד(נמרוד, ספרות)
בין   לומד(X, ספרות)   לבין   לומד(יובל, ספרות)
```

כמו כן התוודענו כי **אין התאמה** בין לומד(X, ספרות) לבין לומד(נמרוד, פרולוג).

דוגמה 10.6

נעיין בביטוי select(X, [X|T], T) ובביטוי select(X, [1,2,3], Y). יש התאמה בין הביטויים האלה:

בין	[X|T]	לבין	[1,2,3]
בין	X	לבין	1
בין	T	לבין	[2,3]
בין	T	לבין	Y
בין	select(X, [X|T], T)	לבין	select(X, [1,2,3], Y)

המאורע המשותף של שני הביטויים האחרונים הוא

```
select(1, [1,2,3], [2,3])
```

דוגמה 10.7

בין הביטוי X לבין f(X) יש התאמה. ההצבה אשר בעקבותיה הם זהים היא:

ביטוי		משתנה
f(X)	=	X

את f(X) מציבים תחת X בביטוי הראשון.
(נציין כי ההצבה X=f(X) אסורה בפרולוג מטעמים, אשר בהם נדון בסעיף הבא).

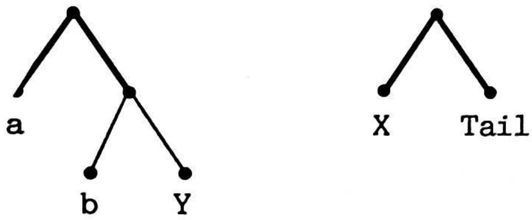
את דבר ההתאמה בין שני הביטויים Term1 ו-Term2 קובעים על פי הכללים האלה:

- אם הביטויים Term1 ו-Term2 קבועים, יש ביניהם התאמה רק אם הם זהים תחבירית.
- אם לפחות אחד מהם, למשל Term1, הינו משתנה, אז יש ביניהם התאמה. להלן ההצבה אשר לאחריה הביטויים יהיו זהים:

ביטוי		משתנה
Term2	=	Term1

- אם Term1 ו-Term2 שניהם ביטויים מורכבים, אז יש ביניהם התאמה רק כאשר:
  - הפנקטור הראשי של Term1 זהה לזה של Term2 וגם -
  - האריטי של Term1 זהה לזה של Term2 וגם -
  - יש התאמה בין הארגומנטים: בין ראשון לראשון, בין שני לשני וכך הלאה.

את ההתאמה בין שני ביטויים מורכבים נוח לבצע כאשר כל אחד מהביטויים מיוצג בצורת עץ. השורש הוא הפנקטור הראשי של הביטוי ומספר בניו הם האריטי שלו. אם השורש ומספר בניו של שני הביטויים זהים, יורדים למפלס הבא ובודקים אם קיימת התאמה בין הבנים. נתבונן באיור:



איור 10.3 – הצגה של הביטויים [X|Tail] ו-[a,b|Y] בצורת עץ לצורך התאמה

בין הביטוי [X|Tail] לבין [a,b|Y] יש התאמה. ההצבה אשר מזהה אותם היא (בתעתיק לועזי):

משתנה		ביטוי
X	=	a
Tail	=	[b|Y]

10.4 איחוד

נחזור אל הפיסקאות a ו-b של דוגמה 10.2. בפיסקאות אלו מופיעים משתנים. המשתנה הלוגי מייצג עצם או ביטוי בלתי ידועים. טווח המשתנה הוא פיסקה אחת בלבד ולכן לא מן הנמנע הוא, שאותם משתנים יופיעו בשתי הפיסקאות. נניח כי בשל סיבה לא ידועה השתמשנו במשתנים U,Z,Y,X בתוך הפיסקאות a ו-b באופן הבא:

בפיסקה a: משפחה(X, אם(צבר, Y), Z).
בפיסקה b: משפחה(אב(X, Y), Z, U)

תהליך ההתאמה המופעל באופן פורמלי בין הפיסקאות a ו-b יוביל אותנו אל התוצאות האלה:

הצבה θ1: X = אב(X, Y)
1Z = Z
הצבה θ2: Z = אם(צבר, Y).
U = 1Z.

אחרי ביצוע ההצבות θ1 ו-θ2 נקבל את המאורע המשותף הבא:

משפחה(אב(X, Y), אם(צבר, Y), 1Z)

משתמע מכך, כי השם הפרטי של האב זהה לזה של האם, אך זו לא היתה כוונתנו. המשתנה Y של פיסקה a לא היה קשור אל המשתנה Y של פיסקה b, כדוגמת המשתנה X בשתי פיסקאות האלה (ראה תכנית 7.2 – "תלמידים" שבפרק 7):

```
סופר(X):- לומד(X, ספרות).
איש_מחשבים(X):- לומד(X, מתמטיקה), לומד(X, פרולוג).
```

הזהות בין השם הפרטי של האב לבין זה של האם אירעה בשל **התנגשות בין שמות המשתנים** (name clash). המשתנה Y, שהופיע בשתי הפיסקאות במקום השמות הפרטיים, הוביל אותנו אל מסקנות שגויות. הבעיה הנוספת בשתי הפיסקאות האלו היתה הופעת משתנה X בשני מפלסים שונים: במקום הביטוי המורכב אב / 2 בפיסקה הראשונה, וכארגומנט של הפרדיקט אב בפיסקה השניה. על כך נדון בהמשך (ראה גם דוגמה 10.7 ותרגיל 10.1).
כדי למנוע את ההתנגשות, הקפדנו באופן אינטואטיבי לסמן את משתני הפיסקאות a ו-b בדוגמה 10.2 באותיות שונות. אבל המחשב פועל על פי האלגוריתם ולא באופן אינטואטיבי ולכן תהליך ההתאמה זקוק לשיפור שימנע את התנגשות שמות המשתנים.
שיפור תהליך ההתאמה מביא אותנו אל **תהליך האיחוד** (unification). תהליך זה תואר לראשונה ב-[Robinson, 1965]. ישנם אלגוריתמים שונים של איחוד. להלן נתאר באופן בלתי פורמלי אחד מהם.
האלגוריתם מתבצע בין שני ביטויים Term1 ו-Term2, אשר אותם יש לאחד. כתוצאת ביצוע האלגוריתם מתקבלת הצבה θ, אשר מגדירה את המאורע המשותף הכללי ביותר של Term1 ו-Term2, או מתקבלת הודעת כשלון (failure) אם אין הצבה כזו.
האלגוריתם פועל על פי הכללים האלה.

כלל 1: קבוע – קבוע

המקרה הפשוט ביותר הוא כאשר Term1 ו-Term2 הינם קבועים. אם הם זהים, אז הם גם המאורע המשותף הכללי ביותר שלהם. ההצבה במקרה זה היא הצבה ריקה (לפעמים מכנים אותה בשם nil). אם הקבועים אינם זהים מתקבלת הודעת כשלון.

כלל 2: משתנה – משתנה אחר

מקרה פשוט אחר הוא כאשר Term1 ו-Term2 הינם משתנים אשר סימוניהם שונים (למשל, האחד – X והשני – Y). במקרה זה מחליפים את הסימון (למשל, את שניהם מסמנים ב-Z1, ובמקרה זה ההצבה θ נראית בתעתיק לועזי משמאל לימין, כך: (X=Z1, Y=Z1).

כלל 3: ביטוי מורכב – ביטוי מורכב

נניח כי Term1 ו-Term2 הם ביטויים מורכבים. אם הפנקטור הראשי של הביטוי האחד שונה מזה של השני, או אם האריטי של האחד שונה מזה של השני, מתקבלת הודעת כשלון. את הפנקטור והאריטי של האחד זהים לאלה של השני, מתבצעת פעולת איחוד בין הארגומנטים. האיחוד מתבצע בין הארגומנטים הנמצאים במקומות זהים בכל אחד מהביטויים: בין הארגומנט הראשון של אחד הביטויים לבין הארגומנט הראשון של הביטוי האחר, בין הארגומנט השני של הביטוי האחד לבין הארגומנט השני של הביטוי השני וכך הלאה. על פי עקרון התכנות הלוגי, ניתן להתחיל את האיחוד ממקום כלשהו ולאו דווקא מן המקום הראשון.
אם כן, פעולת האיחוד היא פעולה החוזרת על עצמה, משמע – פעולה רקורסיבית. את הכלל האחרון של פעולת האיחוד נתאר כצעד בתוך הלולאה.

כלל 4: משתנה – ביטוי

הכלל האחרון מתאר את המקרה כאשר לפחות אחד מהביטויים הוא משתנה. כלל 2, אשר אותו הצגנו בנפרד כדי להקל על הקורא, הוא המקרה הפרטי של כלל 4. את המשתנה הזה נסמן ב-X, ואת הביטוי, איתו X מתאחד, נסמן ב-term. הצעד הראשון של פעולת האיחוד הוא **בדיקת הנוכחות** (occurs check):

בודקים האם X מופיע ב-term

אם X אכן מופיע בתוך term (למשל אם term=f(X) כמו בדוגמה 10.7), מתקבלת הודעת כשלון. אם X איננו מופיע ב-term- **נעשית פעולת הצבה של term תחת X בכל אחת מהופעותיו של X.**

מהן ההופעות של X אשר בהן מתבצעת ההצבה? ראשית, מכיון שאנחנו נמצאים בתוך לולאת האיחוד ונעשו כבר הצבות תחת משתנים אחרים, עלול X להופיע באותן הצבות. שנית, X עלול להופיע בארגומנטים האחרים של שני הביטויים אשר אותם אנחנו מאחדים.

ניעזר בדוגמה. נניח כי האיחוד בין X לבין term=[b] הוא חלק של תהליך של איחוד הביטויים:

```
append(A, X, [a,b|X]),          append([a|X],[b],Y).
```

הפנקטור והאריטי של הביטוי האחד זהים לאלו של האחר. המשתנה X והביטוי [b] הינם הארגומנט השני של append/3, ואנו רוצים לאחדם. נניח כי לפני כן התבצע האיחוד בין A לבין [a|X]. מכאן, שהופעותיו של X הן:

משתנה ביטוי

A = [a|X] – בתוך ההצבה

– בתוך הארגומנט האחרון [a,b|X] של אחד הביטויים.

על כן, יש להציב את [b] תחת X בכל אחד מהביטויים: [a|X], [a,b|X].

לאחר הצבת term תחת המשתנה X בכל אחת מהופעותיו, המשתנה X "מחוסל". כתוצאה של ביצוע מוצלח של האיחוד מתקבלת ההצבה θ:

X1 = term1, X2 = term2, ..., XN = termN

כאשר אף לא משתנה אחד מתוך X1, X2, ..., XN מופיע בביטוים term1, term2, ..., termN. נציין, כי ביישומים של אלגוריתם האיחוד אין מבצעים את ההצבות תחת המשתנים (כלומר, אין מחליפים את תכולת תאי הזכרון), אלא מציינים את הקשר בין כתובות של המשתנים והביטוי שמחליף אותו.

דוגמאות של איחוד

דוגמה 10.8

נאחד באמצעות האלגוריתם דלעיל את הביטויים שלהלן:

```
Term1 = append(A, X, [a, b|X]).
Term2 = append([a|X], [b], Y).
```

התחלה: הביטויים Term1 ו-Term2 הינם ביטויים מורכבים. לשניהם מבנה אחיד append/3 ועל כן בהתאם לכלל 3, נעבור לאיחוד הארגומנטים. אלגוריתם האיחוד אינו מפרט את סדר בדיקת הארגומנטים ולכן נאחד אותם לפי הסדר הבא: השני, הראשון והשלישי.

איחוד הארגומנטים: הארגומנט השני של Term1 הוא המשתנה X. הארגומנט השני של Term2 הוא הביטוי המורכב [b]. בהתאם לכלל 4 מבצעים את הצעדים הבאים.

בדיקת הנוכחות: המשתנה X אינו מופיע בתוך הביטוי [b].
הצבה: נציב את [b] תחת X בכל אחת מהופעותיו של X ונקבל:

```
(ההצבה)            X = [b]
(עדכון של הביטוי)  Term1 = append(A, [b], [a, b, b]).
(עדכון של הביטוי)  Term2 = append([a, b], [b], Y).
```

הארגומנט הראשון של Term1 הוא המשתנה A והארגומנט הראשון של Term2 הוא הביטוי המורכב [a,b]. בהתאם לכלל 4, מבצעים את בדיקת הנוכחות ואת ההצבה הבאה: A = [a,b]. הארגומנט השלישי של Term1 הוא הביטוי המורכב [a,b,b] והארגומנט השלישי של Term2 הוא המשתנה Y. בהתאם לכלל 4, מבצעים את בדיקת הנוכחות ואת ההצבה: Y = [a,b,b].

סיום: בתום ביצוע האיחוד של הביטויים Term1 ו-Term2 מתקבלת ההצבה θ:

X = [b], A = [a,b], Y = [a,b,b]

דוגמה 10.9

נאחד את הביטויים הבאים באמצעות האלגוריתם שלמדנו:

```
Term1 = select(A, [1, 2, 3], L)
Term2 = select(X, [X|T], T)
```

התחלה: הביטויים Term1 ו-Term2 הינם ביטויים מורכבים בעלי מבנה אחיד select/3. על פי כלל 3, נעבור לבדיקת הארגומנטים.

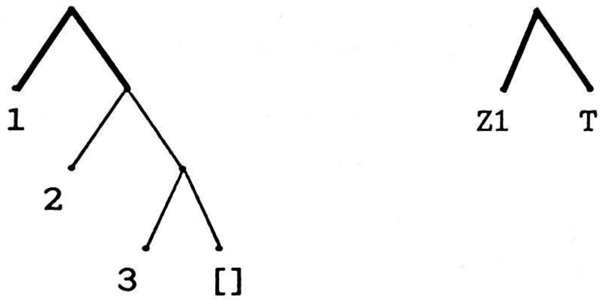
איחוד הארגומנטים: את הארגומנט הראשון מאחדים על פי כלל 2, כלומר מבצעים את ההצבה של Z1 תחת A ותחת X:

```
A = Z1
X = Z1
```

נקבל:

```
(עדכון הביטוי)    Term1 = select(Z1, [1, 2, 3], L)
(עדכון הביטוי)    Term2 = select(Z1 ,[Z1|T], T)
```

הארגומנט השני של Term1 המעודכן הוא הביטוי המורכב [1,2,3], והארגומנט השני של Term2 המעודכן, הוא הביטוי המורכב [Z1|T].
האיור 10.4 מציג את כל אחת מהרשימות [1,2,3] ו-[Z1|T] בצורת עץ:



איור 10.4 – הצגת הרשימות בצורה של עץ לצורך האיחוד

הרשימות [1,2,3] ו-[Z1|T] הן בעלות מבנה זהה, עם פנקטור "דוט" (נקודה) ואריטי 2, ולכן ניתן על פי כלל 3 לעבור לאיחוד הארגומנטים. את הארגומנטים מאחדים על פי כלל 4 ומקבלים את ההצבה הבאה:

```
Z1 = 1
T = [2,3]
```

נבצע הצבה זו בכל המופעים של Z1 ושל T בהצבה הקודמת ובביטויים Term1 ו-Term2 המעודכנים:

```
A = 1
X = 1
Term1 = select(1, [1, 2, 3], L)
Term2 = select(1, [1, 2, 3], [2, 3])
```

(עדכון נוסף של הביטוי)
(עדכון נוסף של הביטוי)

את הארגומנט השלישי מאחדים על פי כלל 4: מבצעים את בדיקת הנוכחות ואת ההצבה L=[2,3].

סיום: בתום ביצוע האיחוד של הביטויים Term1 ו-Term2 מתקבלת ההצבה θ:

```
A = 1, X = 1, T = [2,3], L = [2,3]
```

נזכיר כי אלגוריתם האיחוד אינו מפרט את סדר בדיקת הארגומנטים. סדר זה נקבע בעת היישום.

ההצבה שמתקבלת בסיום תהליך האיחוד בין שני הביטויים Term1 ו-Term2 נקראת **המאחד הכללי ביותר** (most general unifier, או בראשי תיבות mgu) של הביטויים Term1 ו-Term2.

ה-mgu מגדיר את **המאורע המשותף הכללי ביותר** של הביטויים Term1 ו-Term2.

המאורע המשותף הכללי ביותר של שני הביטויים המוגדר באמצעות ה-mgu כולל לפעמים גם משתנים. נציג זאת בדוגמה.

דוגמה 10.10

נאחד את הביטויים האלה:

```
Term1 = record(student(FirstName, LastName),
                    faculty(Faculty),
                    studies(List_of_courses))
Term2 = record(X, faculty(computer_sciences),
               studies([pascal, prolog, data_structure])
```

בתום האיחוד נקבל את ה-θ mgu:

```
X = student(FirstName, LastName)
Faculty = computer_sciences
List_of_courses = [pascal, prolog, data_structure].
```

המאורע המשותף הכללי ביותר של Term1 ו-Term2, המוגדר באמצעות ה-θ, mgu הוא:

```
record(student(FirstName, LastName),
              faculty(computer_sciences),
              studies([pascal,prolog,data_structure])).
```

מאורע זה כולל משתנים. ניתן לפרש כך: כל סטודנט בפקולטה למדעי המחשב לומד prolog, pascal ו-data_structure.

ההצבה, אשר כתוצאת ביצועה יהיו הביטויים Term1 ו-Term2, זהים מבחינה תחבירית, מכונה בשם **מאחד** (unifier). המאחד מגדיר את המאורע המשותף של הביטויים.

באשר להבדל בין התאמה לבין איחוד: בדיקת הנוכחות היא אשר מבדילה בין שתי הפעולות האלו. ברוב הגירסאות של פרולוג בדיקת הנוכחות אינה מיושמת מטעמי היעילות, אולם לפני התחלת האיחוד נעשית החלפת שמות המשתנים, שמונעת את התנגשות השמות בעת ביצוע הפעולה.

תרגיל 10.1

א. מצא את ה-mgu של הביטויים הבאים:

```
unify([a,b,c], A, L)          unify([X|Y], Y, [X|A])
```

ב. מצא מה תהיה התשובה לשאילתה המורכבת בהתאם לגירסת פרולוג אשר ברשותך. תן הסבר לתשובה.

```
?- unify([a,b,c], A, L), unify([X|Y], Y, [X|A]).
```

ג. הצגנו את השאילתה:

```
?- X=f(X).
```

וקיבלנו את התשובה הבאה (בגירסת WISDOM):

```
X = f(f(f(f(f(f(f(f(f(f(f(f(f(
[Out of stack]
```

תן הסבר לתשובה זו.

ד. הצגנו את השאילתה:

```
?- X=אב(Y,X)
```

מה לפי מיטב הבנתך תהיה התשובה הצפויה של פרולוג? נמק.

ה. בדוק מה תהיה התשובה לשאילתה הבאה על פי גירסת פרולוג שברשותך:

```
?- unify(X, f(X)), unify(Y, f(f(Y))), unify(X, Y).
```

הערה: השאילתה האחרונה מוצגת במאמר [Filgueiras, 1984].

10.5 המודל החישובי של תכנית לוגית

כאשר נתונה תכנית לוגית, אנחנו מציגים שאילתות ומקבלים תשובות. התשובה מתקבלת בתום **תהליך החישוב** (computation).

בסעיף זה נציג את **המודל החישובי** (computational model) של תכנית לוגית. ניעזר במונחים הבאים:

* **מטרה** (goal)- שאילתה שהיא נוסחה אטומית, כלומר אינה שאילתה מורכבת.
* **הורדת המטרה** (goal reduction)- המרת המטרה בגוף החוק, אשר ראשו מתאחד עם המטרה (ניתן גם להשתמש במונח **צימצום המטרה**).
* **המטרה הנגזרת** (derived goal)- מטרה חדשה שנלקחה מגוף החוק, אשר החליף את המטרה הקודמת.

נתאר את המונחים האלה באמצעות האיור 10.5:

תכנית לוגית :PROGRAM	a:-b. b:-c.
שאילתה:	?-a.
מטרה:	a
הורדת המטרה:	| :-b
מטרה:	b
הורדת המטרה:	| :-c
מטרה:	c

איור 10.5 – הורדת המטרה

המטרה b שבאיור 10.5, היא המטרה הנגזרת, וכך גם המטרה c.

* **רזולבנטה** (resolvent)- מטרה, או קבוצת המטרות הקשורות באמצעות הקשר AND. בספרי לימוד רבים, כמו למשל ב-[Nillson, 1980], מכנים בשם זה את תוצאות הפעלתו של כלל היסק. לדוגמה, הפסוק $a \vee c$ הינו רזולבנטה של הפסוקים $\{a \vee (\neg b), b \vee c\}$ על פי כלל ההיסק resolution. הרזולבנטה של תכנות לוגי מתקבלת כתוצאה של הורדת המטרה באמצעות כלל ההיסק modus ponens.

* **רזולבנטה התחלתית** – שאילתה, פשוטה או מורכבת.

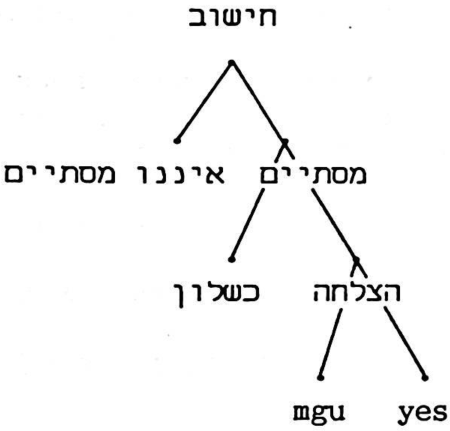
נתאר את הרזולבנטה באמצעות התרשים שבאיור 10.6.

תכנית לוגית :PROGRAM	a:-b,c. b:-b1,b2.
שאילתה:	?- a.
רזולבנטה (התחלתית)	a
	|
רזולבנטה	b,c
	|
רזולבנטה	b1,b2,c

איור 10.6 – רזולבנטה

* **מטרה בסיסית** (ground goal)- מטרה שאיננה כוללת משתנים.

* **חישוב של תכנית לוגית** – תהליך קבלת תשובה לשאילתה.
  - **התחלת החישוב**: החישוב מתחיל כאשר מתקבלת שאילתה, פשוטה או מורכבת.
  - **סיום החישוב**: החישוב מסתיים, או שאיננו מסתיים.
  - **תוצאת החישוב אשר הסתיים**: הצלחה או כשלון. במקרה של כשלון מתקבלת התשובה no. במקרה של הצלחה מקבלים:
    - **תשובה** yes, אם השאילתה היתה בסיסית, או כל אחד ממשתני השאילתה היה משתנה אנונימי.
    - **ה-**mgu של השאילתה ושל סדרת המטרות, אשר נגזרו מהרזולבנטה ההתחלתית באמצעות ההצבות על פי כלל ההיסק modus ponens.



איור 10.7 – חישוב של תכנית לוגית

להלן התאור של המודל החישובי של תכנית לוגית. את צעדי החישוב נתאר בשפה דמויית בייסיק:

א. **התחלת החישוב**: תכנית לוגית PROGRAM ושאילתה R, אשר היא גם הרזולבנטה ההתחלתית.

ב. **צעדי החישוב**:

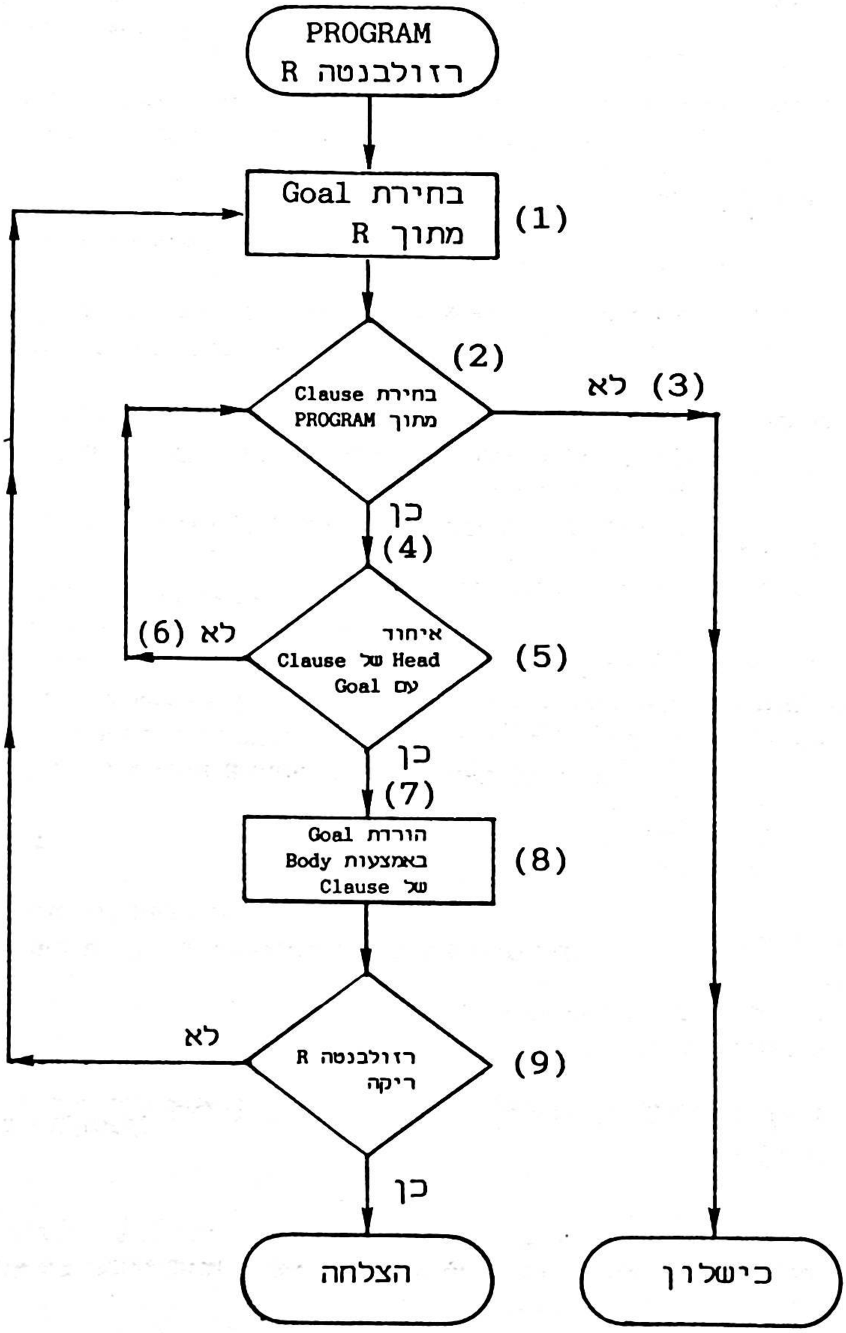
1. מתוך הרזולבנטה R בוחרים את המטרה Goal.
2. מתוך התכנית PROGRAM בוחרים את הפיסקה Clause, בעלת ראש Head, אשר הפנקטור הראשי והאריטי של Head זהים לאלו של המטרה Goal.
3. אם אף לא פיסקה אחת ב-PROGRAM הינה בעלת ראש Head, אשר הפנקטור הראשי שלו והאריטי זהים לאלו של Goal, עוברים לסיום א׳, אחרת, עוברים לצעד 4.
4. מחליפים את שמות משתני הפיסקה Clause כדי למנוע את ההתנגשות בינם לבין משתני הרזולבנטה R. (את הצורה האלפאביתית של הפיסקה נמשיך לסמן ב-Clause).
5. מאחדים את המטרה Goal עם הראש Head של הפיסקה Clause. בזמן האיחוד מקפידים על בחירת שמות המשתנים כדי שיהיו שונים מאלה אשר הופיעו ברזולבנטה R, בפיסקה Clause ובצעדים הקודמים.
6. אם תהליך האיחוד מסתיים בכשלון, עוברים לצעד 2. אחרת, עוברים לצעד 7.
7. את ה-mgu θ של המטרה Goal ושל הראש Head מציבים בתוך הפיסקה Clause ובתוך כל אחת מהמטרות של הרזולבנטה R (את המאורע של Clause נמשיך לסמן ב-Clause, ואת המאורע של R נמשיך לסמן ב-R).
8. מבצעים את הורדת המטרה Goal על ידי המרתה בגוף Body של הפיסקה Clause. את הרזולבנטה החדשה אשר נקבל, נמשיך לסמן ב-R.
9. אם הרזולבנטה R ריקה, עוברים לסיום ב׳, אחרת – עוברים לצעד 1.

ג. **סיום החישוב:**

סיום א': החישוב הסתיים בכשלון.

סיום ב': החישוב הסתיים בהצלחה.

התרשים שבאיור 10.8 מתאר את צעדי החישוב של תכנית לוגית. המספרים שבסוגריים מסמנים את הצעדים שהוסברו קודם.



איור 10.8 – מודל חישובי של תכנית לוגית

נדגים את המודל החישובי של תכנית לוגית.

דוגמה 10.11

א. **התחלת החישוב**: תכנית לוגית Dictionary (ראה פרק 5), ושאילתה מורכבת:
?-means(stone, X), begins(X, Y)

ב. **צעדי החישוב:**

1. מתוך הרזולבנטה R: **means(stone,X),begins(X,Y)**, בוחרים את המטרה Goal: **means(stone,X)**

2. מתוך התכנית **Dictionary** בוחרים את הפיסקה Clause: **means(stone,even)** בעלת ראש Head, אשר הפנקטור והאריטי שלה זהים לאלו של Goal.

5. מאחדים את המטרה Goal: **means(stone,X)**, עם הראש Head של הפיסקה Clause: **means(stone,even)**

7. את ה-mgu θ של Goal ושל Head: **X = even**, מציבים בתוך כל אחת מהמטרות של הרזולבנטה R: **means(stone,even), begins(even,Y)**

8. מבצעים את הורדת המטרה Goal על ידי המרתה בגוף Body של הפיסקה Clause. הגוף Body ריק ועל כן הרזולבנטה החדשה R היא: **begins(even,Y)**.

9. הרזולבנטה R: **begins(even,Y)**, אינה ריקה ולכן עוברים לצעד 1:

   1. המטרה Goal: **begins(even,Y)**.
   2. הפיסקה Clause: **begins(even,aleph)**.
   5. מאחדים את המטרה Goal עם הראש Head של הפיסקה Clause.
   7. את ה-mgu: **Y = aleph**, מציבים בתוך המטרה Goal: **begins(even, aleph)**
   8. מורידים את המטרה Goal על ידי המרתה בגוף הריק של הפיסקה Clause.
   9. הרזולבנטה החדשה R ריקה ועל כן עוברים לסיום ב'.

ג. **סיום החישוב:**

סיום ב'. החישוב הסתיים בהצלחה.
תוצאת החישוב. ה-mgu של השאילתה ושל סדרת המטרות:

X = even
Y = aleph

נעיין בדוגמת חישוב נוספת.

דוגמה 10.12

א. **התחלת החישוב**: תכנית לוגית "**תלמידים**" (ראה פרק 5) ושאילתה: ?-איש מחשבים(רמי).

ב. **צעדי החישוב:**

1. המטרה Goal: **איש_מחשבים(רמי).**

2. הפיסקה Clause בעלת ראש Head, אשר מבנהו איש_מחשבים(X):
איש_מחשבים(X):- לומד(X, מתמטיקה), לומד(X, פרולוג).

5. מאחדים את Goal עם Head של Clause.

7. את ה-mgu: X=**רמי** מציבים בתוך Clause:
איש_מחשבים(רמי):-לומד(רמי, מתמטיקה), לומד(רמי, פרולוג).

8. מבצעים את הורדת המטרה Goal על ידי המרתה בגוף Body של הפיסקה Clause.
הרזולבנטה R היא:
לומד(רמי, מתמטיקה), לומד(רמי, פרולוג).

9. הרזולבנטה R איננה ריקה ועל כן עוברים לצעד 1.

   1. מתוך R בוחרים את Goal: **לומד(רמי,פרולוג).**

   2. מתוך התכנית בוחרים את Clause: **לומד(רמי,פרולוג).**

   5. מאחדים את Goal עם Head של Clause.

   8. מורידים את המטרה Goal על ידי המרתה בגוף ריק של Clause. הרזולבנטה R היא : **לומד(רמי,מתמטיקה).**

   9. הרזולבנטה R איננה ריקה ועל כן עוברים לצעד 1.

      1. המטרה: **לומד(רמי,מתמטיקה).**
      2. הפיסקה: **לומד(רמי,מתמטיקה).**
      5. מאחדים.
      8. מורידים.
      9. הרזולבנטה R ריקה.

ג. **סיום החישוב:**

סיום ב'. החישוב הסתיים בהצלחה.
תוצאת החישוב: התשובה "כן".

10.6 יישום של המודל החישובי: בחירת מטרה ובחירת פיסקה

כאשר תיארנו את המודל החישובי של תכנית לוגית, לא פירטנו את שני הצעדים האלה: בחירת המטרה מתוך הרזולבנטה ובחירת הפיסקה מתוך התכנית. תחילה נדון בבחירת המטרה. נזכיר את הצעד המתאים של המודל החישובי:

10.6(א) מתוך רזולבנטה R בוחרים את המטרה Goal.

מזוית ראיה לוגית, הרזולבנטה שכוללת יותר ממטרה אחת היא פסוק הבנוי מפסוקים אטומיים באמצעות הקשר AND. לכן סדר בחירת המטרה איננו חשוב, שהרי הפסוק המורכב a AND b שקול מבחינה סמנטית לפסוק b AND a. אולם, מבחינה מעשית יש חשיבות רבה לסדר בחירת המטרה. על ידי בחירה נבונה ניתן לקצר את תהליך החישוב. במקרה הקיצוני, הבחירה הבלתי נכונה של המטרה גורמת לכך שאי אפשר לסיים את תהליך החישוב.
נעיין בתהליך החישוב של השאילתה הבאה:

אב_בשושלת(אברהם, קובי).

דוגמה 10.13

א. **התחלת החישוב**: התכנית הלוגית "**שושלת היוחסין של קובי**" (ראה סעיף 7.8) והשאילתה **אב_בשושלת(אברהם,קובי).**

ב. **צעדי החישוב:**

1. המטרה Goal היא:
אב_בשושלת(אברהם, קובי).

2. מתוך התכנית בוחרים את הפיסקה Clause:
אב_בשושלת(Y,X):- **אב**(L,X), **אב_בשושלת**(Y,L).

5. מאחדים את Goal עם הראש Head של הפיסקה Clause.

7. ה-mgu θ הוא:
X = **אברהם.**
Y = **קובי.**
את ה-mgu הזה מציבים בתוך הפיסקה Clause ומקבלים:
אב_בשושלת(אברהם,קובי):-אב(אברהם,L**), אב_בשושלת(**L**,קובי).**

8. מורידים את המטרה. הרזולבנטה החדשה R היא:
אב(אברהם,L**), אב_בשושלת(**L**,קובי).**

9. הרזולבנטה R אינה ריקה ועל כן עוברים לצעד 1. אין המודל מפרט איזו מהמטרות יש לבחור, לכן:

1. **מתוך R בוחרים את המטרה Goal: אב_בשושלת(**L**, קובי).**

2. מתוך התכנית בוחרים את הפיסקה Clause:
אב_בשושלת(Y,X):- **אב**(L,X), **אב_בשושלת**(Y,L).

4. מחליפים את שמות משתני הפיסקה Clause:
אב_בשושלת(1Y,1X):- **אב**(1L,1X), **אב_בשושלת**(1Y,1L).

5. מאחדים את המטרה Goal עם הראש Head של Clause.

7. ה-mgu θ של Goal ושל Head הוא:
L = 1X
1Y = **קובי.**
מציבים אותו בתוך Clause ומקבלים:
אב_בשושלת(L,**קובי**):-**אב**(1L,L), **אב_בשושלת**(1L,**קובי**).

8. מורידים את המטרה. הרזולבנטה החדשה R היא:
אב(1L,L), **אב_בשושלת**(1L,**קובי**).

9. הרזולבנטה R אינה ריקה ועל כן עוברים לצעד 1. אין המודל מפרט איזו מהמטרות יש לבחור, ולכן:

1. **מתוך R בוחרים את** Goal: **אב_בשושלת**(1L,**קובי**).
המטרה הנגזרת **אב_בשושלת**(1L,**קובי**) היא הצורה האלפביתית של המטרה הקודמת **אב_בשושלת**(L,**קובי**).
נמשיך את החישוב.

2. מתוך התכנית בוחרים את Clause:
אב_בשושלת(Y,X):- **אב**(L,X), **אב_בשושלת**(Y,L).

4. מחליפים את שמות המשתנים:
אב_בשושלת(2Y,2X):- **אב**(2L,2X), **אב_בשושלת**(2Y,2L)

5. מבצעים את האיחוד

7. ה-mgu הוא:
L = 2X
2Y = **קובי.**
מציבים אותו בתוך Clause ומקבלים:
אב_בשושלת(L,**קובי**):-**אב**(2L,L), **אב_בשושלת**(2L,**קובי**).

8. מורידים את המטרה. הרזולבנטה החדשה R היא:
אב(2L,L), **אב_בשושלת**(2L,**קובי**).

9. הרזולבנטה R איננה ריקה, ועל-כן עוברים לצעד 1. אין המודל מפרט איזו מהמטרות יש לבחור ולכן:

1. **מתוך R בוחרים את** Goal: **אב_בשושלת**(2L,**קובי**).

וחוזר חלילה. נקבל את סדרת הצורות האלפביתית של המטרה **אב_בשושלת**(L,**קובי**). החישוב איננו מסתיים.

```
אב_בשושלת(אברהם,קובי)
        |
        |
        |
אב(אברהם,L), אב_בשושלת(L,קובי)
        |
        |
        |
אב(L,1L), אב_בשושלת(1L,קובי)
        |
        |
        |
אב(1L,2L), אב_בשושלת(2L,קובי)
        |
        |
        |
```

איור 10.9 – בחירה של המטרה מתוך רזולבנטה

את תהליך בחירת המטרה מכנים בשם **מדיניות הקביעה** (scheduling policy). לכל יישום יש מדיניות קביעה האופיינית לו. ביישומי המודל החישובי של תכנית לוגית, שמיועדים למחשב המקבילי, שואפים לחשב בו-זמנית מספר מטרות (and-parallelism), ראה [Shapiro, 1983]. שפת הפרולוג, המיישמת את המודל החישובי של תכנית לוגית במחשב בעל פעולה סדרתית, מאמצת בתור מדיניות הקביעה את **מדיניות המחסנית** (stack policy):

- **הפרולוג האנגלי בוחר מתוך הרזולבנטה R את המטרה השמאלית ביותר.**
- **הפרולוג העברי בוחר מתוך הרזולבנטה R את המטרה הימנית ביותר.**

לאחר הבחירה מתבצעת הורדת המטרה ודחיפת המטרה הנגזרת אל ראש המחסנית. אם כן, מתוך הרזולבנטה R:

אב(אברהם,L),אב_בשושלת(L,קובי).

תיבחר המטרה Goal: **אב(אברהם,L)**. חישוב עם הבחירה הזאת הוצג באופן בלתי פורמלי בסעיף 7.8.

הצעד האחר שאיננו מפורט במודל החישובי של תכנית לוגית הינו בחירת הפיסקה:

10.6(ב) מתוך התכנית PROGRAM בוחרים את הפיסקה Clause בעלת ראש Head אשר הפנקטור הראשי והאריטי של Head זהים לאלו של המטרה Goal.

מזווית ראיה לוגית ניתן לראות את הפיסקאות בעלות ראש Head המתאחד עם Goal, כקשורים באמצעות הקשר הלוגי OR. לדוגמה:

```
append([],L,L).
OR
append([X|T],L,[X|NewT]):-append(T,L,NewT).
```

הפסוק a OR b שקול סמנטית לפסוק b OR a. לכן תיאורטית אין זה משנה אם נתחיל את החישוב מפסוק a, או מפסוק b.
אבל באופן מעשי יש חשיבות רבה לסדר הבחירה של הפיסקה. נעיין בתהליך חישוב השאילתה: append(A,[a,b],B) על פי הפרוצדורה append שבפרק 9.

דוגמה 10.14

א. **התחלת החישוב**: הפרוצדורה append (ראה סעיף 9.3) והשאילתה

```
?- append(A,[a,b],B)
```

ב. **צעדי החישוב:**

1. המטרה Goal היא append(A,[a,b],B). אין המודל מפרט באיזו פיסקה נבחר מתוך הפיסקאות בעלות ראש המתאחד עם המטרה Goal ולכן:

2. **בוחרים מתוך התכנית את הפיסקה** Clause:

```
append([X|T],L,[X|NewT]):-append(T,L,NewT).
```

5. מאחדים את Goal עם הראש Head של Clause.

7. את ה-mgu:

```
A = [X|T]
L = [a,b]
B = [X|NewT]
```

מציבים בתוך המטרה Goal ובתוך הפיסקה Clause. הפיסקה החדשה Clause תהיה:

```
append([X|T], [a,b], [X|NewT]):-append(T,[a,b],NewT)
```

פיסקה זו הינה בעלת ראש Head הזהה תחבירית ל-Goal, שלאחר ההצבה.

8. מבצעים את הורדת המטרה. הרזובלנטה החדשה R (אשר היא גם המטרה הנגזרת) הינה: **append(T, [a,b], NewT)**

9. הרזולבנטה החדשה R אינה ריקה ועל כן עוברים לצעד 1.

1. המטרה Goal: **append(T,[a,b],NewT)**. אין המודל מפרט איזו פיסקה נבחר מתוך הפיסקאות בעלות ראש המתאחד עם Goal ולכן:

2. **מתוך התכנית דלעיל בוחרים את הפיסקה Clause:**

```
append([X|T],L,[X|NewT]):-append(T,L,NewT)
```

4. מחליפים את שמות משתני הפיסקה Clause:

```
append([X1|T1],L1,[X1|NewT1]):-append(T1,L1,NewT1).
```

5. מאחדים את Goal עם הראש Head של פיסקה Clause.

7. את ה-mgu :

```
T = [X1|T1]
L1 = [a,b]
NewT = [X1|NewT1]
```

מציבים בתוך ההצבה הקודמת, המטרה והפיסקה:

```
A=[X,X1|T1]        (ההצבות
B=[X,X1|NewT1]     הקודמות)

append([X1|T1],[a,b],[X1|NewT])        (המטרה)

append([X1|T1],[a,b],[X1|NewT1]):-     (פיסקה)
        append(T1,[a,b],NewT1).
```

8. מבצעים את הורדת המטרה. הרזולבנטה החדה R (שהיא גם המטרה הנגזרת) הינה: **append(T1, [a,b], NewT1)**

9. הרזובלנטה R איננה ריקה, ועל-כן עוברים לצעד 1.

וחוזר חלילה. כל עוד נמשיך לבחור את הפיסקה השניה של התכנית דלעיל, לא נסיים את החישוב. אולם החישוב יסתיים אם נבחר את הפיסקה הראשונה של התכנית:

1. המטרה: **append(T1,[a,b],NewT1)**
2. הפיסקה: **append([],L,L)**
4. החלפת שמות משתני הפיסקה: **append([],NewL,NewL)**
5. מבצעים את האיחוד.
7. את ה-mgu:

```
T1 = []
NewL=[a,b]
NewT1=[a,b]
```

מציבים בתוך ההצבות הקודמות, המטרה והפיסקה:

```
(ההצבות הקודמות)   A=[X,X1]
                   B=[X,X1,a,b]
(המטרה)            append([],[a,b],[a,b]).
(הפיסקה)           append([],[a,b],[a,b]).
```

8. מבצעים את הורדת המטרה.

9. הרזולבנטה החדשה R ריקה ולכן עוברים לסיום ב'.

ג. **סיום החישוב:**

סיום ב': החישוב הסתיים בהצלחה.
תוצאת החישוב: ה-mgu של השאילתה ושל סדרת המטרות:

```
A=[X,X1]
B=[X,X1,a,b]
```

כפי שראינו בדוגמה, סיום או אי-סיום של תהליך החישוב תלוי בבחירת הפיסקה Clause מתוך התכנית PROGRAM. בחירת הפיסקה לצורך הורדת המטרה, במקרים בהם יש יותר מפיסקה אחת, בעלת ראש המתאחד עם המטרה, היא בחירה לא-דטרמיניסטית (nondeterministic). ביישומי תכנות לוגי שנועדו למחשב מקבילי, שואפים להוכיח את השאילתה בו-זמנית במספר דרכים (or-parallelism), ראה: [Shapiro, 1983].
שפת הפרולוג, המיישמת את הרעיון של תכנות לוגי במחשב סדרתי, בוחרת מבין הפיסקאות אשר ראשן מתאחד עם המטרה, את הראשונה לפי סדר ההופעה. הדמיית בחירה לא-דטרמיניסטית נעשית באמצעות **נסיגה** (backtracking).

10.7 יישום של המודל החישובי – תרגול

השאלות שלפנינו הן הדמיית הלא-דטרמיניזם של תכנית לוגית. מטרת התרגול – ללמוד את השפעת הסדר של הפסקאות והמטרות על תהליך החישוב.

שאלה 1

רקע:
לפניך שש גירסאות להגדרת היחס ancestor/2. את הגדרת ancestor1 הצגנו בפרק 7.

```
ancestor1(Ancestor, Person):-
                    parent(Ancestor, Person).
ancestor1(Ancestor, Person):-
                      parent(Ancestor, Link),
                      ancestor(Link, Person).
```

הגדרת ancestor2 היא הדמייה של בחירת המטרה הלא-דטרמיניסטית בעת החישוב של ancestor1.

```
ancestor2(Ancestor, Person):-
                        parent(Ancestor, Person).
ancestor2(Ancestor, Person):-
                       ancestor2(Link, Person),
                       parent(Ancestor, Link).
```

הגדרת ancestor3 היא הדמייה של בחירת הפיסקה הלא-דטרמיניסטית בעת החישוב של ancestor1.

```
ancestor3(Ancestor, Person):-
                        parent(Ancestor, Link),
                        ancestor3(Link, Person).
ancestor3(Ancestor, Person):-
                        parent(Ancestor, Person).
```

הגדרת ancestor4 מיישמת את הכתיבה הלא-דטרמיניסטית של הפסוקים בגוף החוק.

```
ancestor4(Ancestor, Person):-
                        parent(Ancestor, Person).
ancestor4(Ancestor, Person):-
                        ancestor4(Ancestor, Link),
                        parent(Link, Person).
```

הגדרת ancestor5 היא הדמייה של בחירת המטרה הלא-דטרמיניסטית בעת החישוב של ancestor4.

```
ancestor5(Ancestor, Person):-
                    parent(Ancestor, Person).
ancestor5(Ancestor, Person):-
                       parent(Link, Person),
                       ancestor5(Ancestor, Link).
```

הגדרת ancestor6 היא הדמייה של בחירת הפיסקה הלא-דטרמיניסטית בעת החישוב של ancestor4.

```
ancestor6(Ancestor, Person):-
                       ancestor6(Ancestor, Link),
                       parent(Link, Person).
ancestor6(Ancestor, Person):-
                       parent(Ancestor, Person).
```

לפניך בסיס הנתונים:

```
parent(terach, abraham).
parent(abraham, isaac).
parent(isaac, jacob).
parent(jakob, yoseph).
```

בצע את המטלות שלהלן.

א. מצא את כל התשובות לשאילתות האלו :

```
?- ancestor1(Ancestor, Person).
?- ancestor3(Ancestor, Person).
?- ancestor5(Ancestor, Person).
```

ב. תאר את תהליך החישוב של השאילתות האלו :

```
?- ancestor2(terach, yoseph).
?- ancestor4(terach, yoseph).
```

(הערה: ניתן להיעזר ב-trace).

ג. בדוק האם מסתיים חישוב השאילתה:

```
?- ancestor6(abraham, isaac).
```

ד. מצא את כל התשובות לשאילתות האלו. תן הסבר לסדר הופעת התשובות.

```
?- ancestor1(terach,Person).
?- ancestor3(terach,Person).
```

ה. תן הסבר לסדר של הופעת התשובות לשאילתות:

```
?- ancestor1(Ancestor, yoseph).
?- ancestor5(Ancestor, yoseph).
```

ו. בדוק מהן התשובות של פרולוג לשאילתות הבאות:

```
?- ancestor2(Ancestor, Person).
?- ancestor4(Ancestor, Person).
```

תן הסבר לכל אחת מהתשובות שהתקבלו.

ז. באיזו משש התכניות היית בוחר כדי לענות על השאילתה שה-Ancestor בה ידוע, וה-Person לא ידוע, כדוגמת:

```
?- ancestor(abraham, Person).
```

תאר את שיקולי הבחירה.

ח. באיזו תכנית היית בוחר כדי לענות על השאילתה:

```
?- ancestor(Ancestor, jacob).
```

נמק את בחירתך.

שאלה 2

חזור לתרגיל 9.5 של הפרק הקודם וענה על השאלות הבאות:

א. עיין בפרוצדורות sublist1 ו-sublist2 והשווה בין התשובות לשאילתות זהות, אשר התקבלו על-פי הפרוצדורות האלו. תאר את ההבדלים שבין התשובות והסבר אותם.

ב. בצע את המטלה א' כלפי הפרוצדורות sublist3 ו-sublist4.

ג. בצע את המטלה א' כלפי הפרוצדורות sublist5 ו-sublist6.

ד. בצע את המטלה א' כלפי הפרוצדורות sublist7, sublist8 ו-sublist9.

ה. בחלק של השאילתות ישנן תשובות החוזרות על עצמן עם הקשת ";". אתר אותן והסבר את התופעה.

ו. באיזו מן הפרוצדורות היית בוחר כדי לענות על השאלה:

מצא את התת-רשימות של הרשימה הנתונה [a,b,c,d].
נמק את בחירתך.

ז. באיזו מן הפרוצדורות היית בוחר כדי לענות על השאלה:

מצא את הרשימות אשר [a] היא תת-רשימה שלהם.
נמק את בחירתך.

ח. באיזו מן הפרוצדורות היית בוחר כדי לענות על השאילתה:

```
?- sublist([],List).
```

ט. באיזו מן הפרוצדורות היית בוחר כדי לענות על השאילתה:

```
?- sublist(List,List).
```

```
tracing: ¦ ?- double([a,b,c],X).
[0] call: double([a,b,c],X87)?
 [1] call: double([b,c],NewT191).
  [2] call: double([c],NewT272).
   [3] call: double([],NewT353).
   [3] fail: double([],_353).
  [2] exit: double([c],[c,c]).
 [1] exit: double([b,c],[b,b,c,c]).
[0] exit: double([a,b,c],[a,a,b,b,c,c]).
... in 3 reductions.

X = [a,a,b,b,c,c]
```

```
tracing: ¦ ?- double(X,Y).
[0] call: double(X80,Y81)?
 [1] call: double(T177,NewT180)?
  [2] call: double(T263,NewT266)?
   [3] call: double(T349,NewT352)?
    [4] call: double(T435,NewT438)?
     [5] call: double(T521,NewT524)?
      [6] call: double(T607,NewT610)?
       [7] call: double(T693,NewT696)?
        [8] call: double(T779,NewT782)?
         [9] call: double(T865,NewT868)?
          [10] call: double(T951,NewT954)?
```

איור 10.10 – חישוב שתי שאילתות:
(א) החישוב של double([a.b.c], X) מסתיים,
(ב) החישוב של double(X, Y) אינו מסתיים.

```
last(X,[X]).
last(X,[Head¦Tail]) :-
   last(X,Tail).

tracing: ¦ ?- last(a,List).
[0] call: last(a,List81)?
[0] exit: last(a,[a]).
... in 1 reductions.

List = [a] ;
[0] redo: last(a,[a]).
 [1] call: last(a,Tail183).
 [1] exit: last(a,[a]).
[0] exit: last(a,[Head182,a]).
... in 2 reductions.

List = [Head182,a] ;
[0] redo: last(a,[Head182,a]).
 [1] redo: last(a,[a]).
  [2] call: last(a,Tail262).
  [2] exit: last(a,[a]).
 [1] exit: last(a,[Head261,a]).
[0] exit: last(a,[Head182,Head261,a]).
... in 3 reductions.

List = [Head182,Head261,a] .
```

```
last2(X,[H¦T]) :-
   last2(X,T).
last2(X,[X]).

tracing: ¦ ?- last2(a,List).
[0] call: last2(a,List81)?
 [1] call: last2(a,T175)?
  [2] call: last2(a,T259)?
   [3] call: last2(a,T343)?
    [4] call: last2(a,T427)?
     [5] call: last2(a,T511)?
      [6] call: last2(a,T595)?
       [7] call: last2(a,T679)?
        [8] call: last2(a,T763)?
         [9] call: last2(a,T847)?
          [10] call: last2(a,T931)?
```

איור 10.11 – הדמייה של בחירת הפיסקה הלא-דטרמניסטית:
הפרוצדורות last ו-last2.

פרק 11

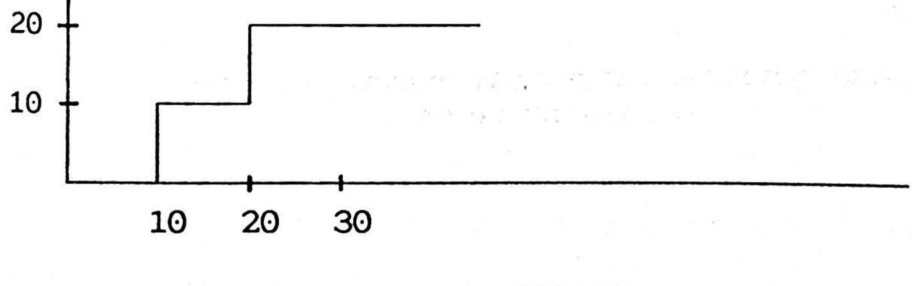
פרדיקטי המערכת Cut ו-not

מערכת של פרולוג מכילה יחסים, הקרויים פרדיקטי המערכת. פרדיקטים אלה הוגדרו מראש והמשתמש או המתכנת אינו יכול לשנותם. בפרק 7 היכרנו אחד מהם, פרדיקט המכונה trace. אלה מן הקוראים, שעובדים עם גירסה WISDOM-PROLOG בוודאי גילו שהיחסים member ו-append הינם פרדיקטי המערכת.

בפרק זה נכיר שני פרדיקטי המערכת: Cut ו-not. הפרדיקט Cut מונע נסיגה וחיפוש של פתרון אלטרנטיבי. את השימוש בו נדגים בסעיף 11.1. הפרדיקט not אמור לבטא את הקשר הלוגי NOT. בסעיף 7.2 הצגנו חוק המכיל את היחס not (לא). בפרק זה נעמיק את ההיכרות עם הפרדיקט not. בסעיף 11.2 נראה איך משתמשים בו בפיסקאות ללא משתנים. בסעיף 11.3 נדון בבעיות שנוצרות עקב השימוש בו בפיסקאות עם משתנים. לסיום, בסעיף 11.4 נציג שאלות לעבודה עצמית בנושאי שימוש בפרדיקטים Cut ו-not.

11.1 בקרת נסיגה באמצעות Cut

מערכת הפרולוג מבצעת בקרה על נסיגה באמצעות פרדיקט המערכת, המסומן ב-"!" והמכונה Cut. את פעולותיו של Cut נכיר באמצעות הדוגמאות שלהלן.



איור 11.1 – מדרגות

דוגמא 1: מדרגות

נעיין במדרגות באיור 11.1. האורך והגובה של כל מדרגה הינם 10 סנטימטרים. באמצעות פונקציה מתמטית ניתן לתאר את המדרגות כך:

$$y(x) = \begin{cases} 0 & x < 10 \\ 10 & 10 \le x < 20 \\ 20 & x \ge 20 \end{cases}$$

היחס שבין ארגומנט הפונקציה x, לבין ערך הפוקנציה y(x), מתבטא על ידי שלושת הכללים האלה:

1. y(x) = 0 IF $x < 10$.
2. y(x) = 10 IF $x \ge 10$ AND $x < 20$.
3. y(x) = 20 IF $x \ge 20$.

את הכללים האלה ניתן לבטא בשפת הפרולוג באמצעות היחס function1(X,Y) אשר בו המשתנה Y מסמן את ערך הפונקציה, התלוי בארגומנט X:

```
function1(X, 0) :- X < 10.
function1(X, 10) :- X >= 10, X < 20.
function1(X, 20) :- X >= 20.
```

תכנית 11.1 – מדרגות

את השאלה **"מהו ערך של Y עבור X=5"** מבטאים באמצעות השאילתה: **?-function1(5, Y)**. נעקוב אחרי תהליך קבלת התשובה:

```
| ?- trace.
tracing: | ?- function1(5, Y).
[0] call: function1(5, Y80)?
 [1] call: 5 < 10?
 [1] exit: 5 < 10?
[0] exit: function1(5, 0)?
... in 1 reductions.

Y = 0
```

התשובה Y=0 התקבלה על פי החוק הראשון. כדי לבדוק אם יש תשובות נוספות נקיש ";". הקשת ";" מתקבלת על ידי מערכת הפרולוג כהודעת כישלון, המחייבת נסיגה.

```
Y = 0;
[0] redo: function1(5,0)?
 [1] redo: 5 < 10?
 [1] fail: 5 < 10?
```

בחיפוש אחר תשובות נוספות, נבדקת האפשרות להיעזר בחוק השני ובחוק השלישי. האפשרויות האלו נפסלות, ולבסוף מתקבלת תשובה שלילית, המעידה על כך שהתשובה היחידה היא Y = 0:

```
 [1] call: 5 >= 10?
 [1] fail: 5 >= 10?
 [1] call: 5 >= 20?
 [1] fail: 5 >= 20?
[0] fail: function1(5, _80)?

          no
```

כדי ליעל את תכנית 11.1 ניעזר בשיקול הבא. הארגומנט X הוא מספר בעל ערך מוגדר. בהתאם לערכו הוא נמצא בתחום השיפוט של חוק אחד בלבד. למשל, אם X=5, אז הוא אינו גדול מ-10, ובוודאי שאינו גדול מ-20. לכן, קבלת תשובה על פי החוק הראשון הינה האפשרות היחידה עבור X<10 ואין צורך לבחון אלטרנטיבות. במילים אחרות, ערכו של הארגומנט X קובע באופן חד-משמעי את החוק שעל פיו מתקבלת התשובה. כדי לחסוך חיפוש מיותר של תשובות אחרות, נוסיף לתכנית 11.1 את פרדיקט המערכת Cut:

```
function2(X, 0) :- X < 10, !.
function2(X, 10) :- X >= 10, X < 20, !.
function2(X, 20) :- X >= 20, !.
```

תכנית 11.2 – מדרגות עם ה-Cut

התכנית 11.2 מבטאת את אותם שלושת הכללים, שביטאה התכנית 11.1, משמע שמבחינה לוגית התכניות 11.1 ו-11.2 הינן שקולות. כדי לעמוד על השינוי הפרוצדורלי, כלומר השינוי בביצוע, נציג שאילתה ונעקוב אחר תהליך קבלת התשובה:

```
| ?- trace.
tracing: | ?- function2(5, Y).
[0] call: function2(5, Y80)?
 [1] call: 5 < 10?
 [1] exit: 5 < 10?
[0] exit: function2(5, 0)?
... in 1 reductions.

Y = 0
```

הקשת ";" מתקבלת על-ידי המערכת של פרולוג כהודעת כישלון, הדורשת לבדוק את האלטרנטיבות. אבל ה-Cut מונע את הנסיגה ובדיקה של אפשרויות נוספות:

```
Y = 0;
[0] redo: function2(5, 0)?
[0] fail: function2(5, _80)?

no
```

החלטנו לערוך שינוי נוסף בתכנית הלוגית המתארת את המדרגות. לשם כך נעזרנו בשיקול הבא: כל עוד X < 10, הפרדיקט Cut מונע לעבור למימוש החוק השני. לכן התנאי X ≥ 10 בחוק השני הינו תנאי מיותר, והוא הדין לגבי התנאי X ≥ 20 בחוק השלישי:

```
function3(X, 0) :- X < 10, !.
function3(X, 10) :- X < 20, !.
function3(X, 20).
```

תכנית 11.3 – מדרגות עם ה-Cut האדום

הפרדיקט Cut שמופיע בתכנית 11.3 מכונה **ה-Cut האדום**. אותו פרדיקט המערכת, המופיע בתכנית 11.2, מכונה **ה-Cut הירוק**. נדון בהבדל שבין שתי הופעותיו של Cut.

התכנית 11.3 פועלת בדומה לתכנית 11.2. נציג את השאלה, מהו הערך של Y עבור X=5. נעקוב אחרי תהליך קבלת התשובה והזמנת תשובות נוספות:

```
| ?- trace.
tracing: ?- function3(5, Y).
[0] call: function3(5, Y80).
 [1] call: 5 < 10?
 [1] exit: 5 < 10?
[0] exit: function3(5, 0)?
... in 1 reductions.

Y = 0;
[0] redo: function3(5, 0)?
[0] fail: function3(5, _80)?

no
```

אפשר ללמוד מכך, כי התכניות 11.2 ו-11.3 מספקות תשובה זהה לשאלה מהו הערך של Y עבור X מסויים. עם הקשת ";" בכל אחת מהתכניות האלו, מתקבלת התשובה no באופן זהה. בכל זאת, יש הבדל מהותי בין התכנית 11.2 לבין 11.3, וכדי להבינו, נחזור שוב לתכנית 11.1. התכניות 11.1 ו-11.2 מספקות תשובות זהות לשאלות זהות. פירוש הדבר, כי אם נמחק את ה-Cut בתכנית 11.2, תשובותיה לא ישתנו. במקרה זה אומרים כי הסמנטיקה (כלומר המשמעות) של תכנית 11.2 איננה משתנה אחרי נשירת ה-Cut. לעומת זאת, אם נמחק את ה-Cut בתכנית 11.3, משמעותה תשתנה. נעיין בתכנית הבאה:

```
function4(X, 0) :- X < 10.
function4(X, 10) :- X < 20.
function4(X, 20).
```

תכנית 11.4 – מחיקת Cut אדום

תחילה נוודא, כי גם על פי תכנית 11.4 הערך של Y עבור X=5 הינו 0.

```
| ?- trace.
tracing: | ?- function4(5, Y).
[0] call: function4(5, Y80)?
 [1] call: 5 < 10?
 [1] exit: 5 < 10?
[0] exit: function4(5, 0)?
... in 1 reductions.

Y = 0
```

כעת נבדוק האם Y = 0 היא התשובה היחידה. נזמין תשובות נוספות:

```
Y = 0;
[0] redo: function4(5, 0)?
 [1] redo: 5 < 10?
 [1] fail: 5 < 10?
 [1] call: 5 < 20?
 [1] exit: 5 < 20?
[0] exit: function4(5, 10)?
... in 1 reductions.

Y = 10;
[0] redo: function4(5, 10)?
 [1] redo: 5 < 20?
 [1] fail: 5 < 20?
[0] exit: function4(5, 20)?
... in 1 reductions.

Y = 20;
[0] redo: function4(5, 20)?
[0] fail: function4(5, _80)?

no
```

את אשר קיבלנו נפרש כך: כתוצאה של הקשת ";" אחרי התשובה Y=0, נסוגה התכנית צעד אחד. היא חוזרת אל המטרה function4(5,Y) ומנסה להפעיל את החוק השני. היא מצליחה לממשו כי הרי 5 קטן מ-20. כך אנחנו מקבלים תשובה נוספת: Y=10. לאחר הקשה נוספת של ";" אנו מקבלים תשובה: Y=20, שהרי העובדה function4(X,20) היא עובדה כללית (ראה סעיף 5.7), המתקיימת עבור כל הצבה תחת X, ובכלל זה עבור X=5.

קיבלנו, כי עבור X=5 על פי התכנית 11.4 יש ל-Y שלושה ערכים שונים. לכן אנו מסיקים, כי היחס function4(X,Y) אינו מבטא את פונקצית המדרגות y(x), כפי שהגדרנו אותה באמצעות שלושת הכללים והיחס function1(X,Y).

כאשר הנוכחות או אי הנוכחות של Cut איננה משנה את משמעות התכנית, אומרים כי ה-Cut הוא **ירוק**. לדוגמה, כל אחד מפרדיקטי ה-Cut של תכנית 11.2 הוא ירוק. כשהתכנית שמכילה Cut שונה מזו המתקבלת אחרי נשירתו, אומרים כי ה-Cut הינו **אדום**. כל אחד מפרדיקטי ה-Cut בתכנית 11.3 הינו אדום.

דוגמה 2: חבר ברשימה

נציג דוגמה נוספת של ה-Cut האדום. נעיין בפרוצדורה הבאה:

```
member1(X, [X|T]) :- !.
member1(X, [Y|T]) :- member1(X, T).
```

תכנית 11.5 – פרוצדורה member עם Cut

נשווה בין תכנית 11.5 לבין הפרוצדורה member של סעיף 9.2. התשובה לשאילתה member1(2, [1,2,3]) תהיה חיובית, כלומר, זהה לזו של member(2,[1,2,3]). הבה נעיין בשאילתה נוספת ובתשובה שהתקבלה:

```
?- member1(X, [1,2,3]).
X = 1;
no
```

ה-Cut מונע קבלת תשובות נוספות. אחרי נשירתו מתקבלת פרוצדורה member שאותה היכרנו בפרק 9. היא שונה מ-member1, כפי שנראה זאת באמצעות השאילתה שלהלן:

```
?- member(X, [1,2,3]).
X = 1;
X = 2;
X = 3;
no
```

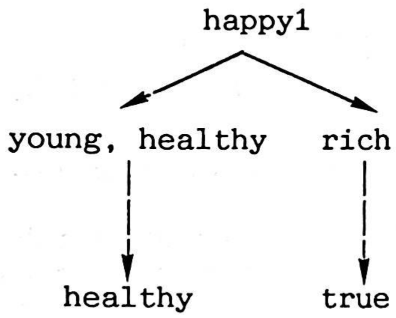
דוגמה 3: הפרוצדורה happy

נעיין בשתי התכניות שלהלן וננתח את התשובות שהתקבלו על פיהן.

happy1 :- young, healthy. happy1 :- rich. young. rich. ?- happy1. yes	happy2 :- young, !, healthy. happy2 :- rich. young. rich. ?- happy2. no

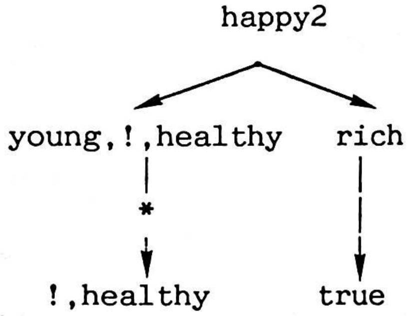
תכנית 11.6 – פרוצדורות happy עם ובלי Cut

את קבלת התשובה לשאילתה **happy1-?** נתאר באמצעות עץ שבאיור 11.2. השאילתה נמצאת בשורשו של העץ. בניו של השורש הינם גופיהם של החוקים, אשר השאילתה עומדת בראשם. הבנים מסודרים משמאל לימין בהתאם לסדר הופעת החוקים בתכנית. בהוכחת השאילתה אנו יורדים לעומק מן הצד השמאלי, עד שמגיעים לעלה healthy, ומשם חוזרים אל השורש ופונים אל הצומת rich. העובדה rich נמצאת בתכנית, מה שמנמק את הופעת עלה true. העץ שבאיור 11.2 הינו **עץ החיפוש** של המטרה happy1. אם כן, התשובה החיובית לשאילתה happy1-? מתקבלת על פי החוק השני: rich :- happy1.



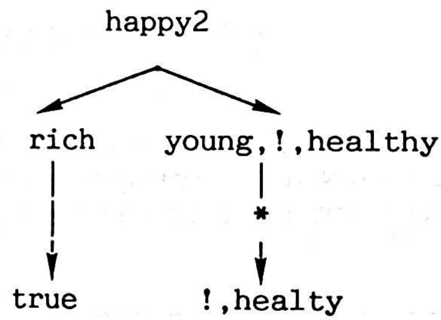
איור 11.2 – עץ החיפוש של happy1

האיור 11.3 מתאר את עץ החיפוש של המטרה happy2. הפרדיקט Cut קוטע את העץ במקום המסומן ב-*. פירוש הדבר הוא, שבהגיעו אל העלה healthy,! אין אפשרות לסגת ולחפש דרך אלטרנטיבית אשר מובילה אל הפתרון. לכן, התשובה לשאילתה **happy2-?** הינה שלילית.



איור 11.3 – קטיעת עץ החיפוש באמצעות Cut

הפרדיקט Cut שמופיע בחוק happy2, הינו אדום. נציין הבדל נוסף בין הפרוצדורות happy1 ו-happy2. שינוי סדר החוקים בפרוצדורה happy1, לא יגרום לשינוי התשובה. במובן זה הפרוצדורה happy1 היא הצהרתית ותואמת את העקרון של התכנות הלוגי. לעומת זאת, השינוי בסדר הופעת החוקים בפרוצדורה happy2 גורם לשינוי התשובה.



איור 11.4 – שינוי סדר החוקים בפרוצדורה happy2

כאשר משנים את סדר החוקים, מגיעה התכנית אל הצומת השמאלי rich, במהלך חיפוש התשובה. משם היא יורדת לעומק, אל העלה true, אשר פירושו – קבלת תשובה חיובית.

ברוב התכניות הלוגיות, שאותן ראינו, שינוי סדר הפיסקאות לא שינה את אוסף התשובות שאותן ניתן היה לקבל. במילים אחרות, שינוי סדר הפיסקאות לא שינה את הסמנטיקה (כלומר את המשמעות) של התכנית. לעומת זאת גילינו, כי כאשר בתכנית מופיע ה-Cut האדום, שינוי סדר הפיסקאות בתכנית עלול לשנות את משמעות התכנית, כלומר – לשנות את אוסף התשובות שהיא מציגה.

סיכום

נתאר את פעולת הפרדיקט Cut באופן כללי. נתבונן בשתי הפיסקאות הבאות:

```
Goal :- Condition1, Condition2, !,
            Condition3, Condition4.
Goal :- Condition.
```

כאשר מציגים את השאילתה **?-Goal** יפנה פרולוג אל החוק הראשון. אם המטרה **Condition1** נכשלה, הוא עובר לחוק השני. אם המטרה Condition1 מצליחה, הוא עובר ל-**Condition2**. אם המטרה Condition2 נכשלה, חוזר פרולוג אל Condition1 ומחפש דרך אלטרנטיבית למלא את התנאי Condition1. אם אין אלטרנטיבות הוא נסוג ל-Goal ופונה אל החוק השני. השפעת ה-Cut אינה מורגשת בשלב זה.
נניח כי המטרה Condition2 מצליחה. הצלחתה של Condition2 גוררת מיידית את הצלחת ה-Cut. עתה עובר פרולוג אל **Condition3**. אם Condition3 נכשלת, נכשלת גם המטרה Goal והמשתמש מקבל הודעת כישלון. אם Condition3 מצליחה, פרולוג עובר אל **Condition4**. נציין כי במקרה של כישלון ב-Condition4, ניתן לסגת מ-Condition4 אל Condition3 ולחפש אלטרנטיבות. לא ניתן להמשיך ולסגת אל Condition2 כי ה-Cut מונע זאת.

בחוקי פרולוג, אשר צורתם: **Goal :- !.**, הפרדיקט Cut מצליח, כאשר מתבצע איחוד בין Goal לבין פיסקה של תכנית, כפי שראינו בפרוצדורה member1. במקרה של member1 נקטע המעבר אל החוק השני בעל ראש זהה, ולכן לא התקבלו תשובות נוספות.

11.2 שלילה באמצעות הכישלון

את השלילה מבטאים בפרולוג באמצעות פרדיקט המערכת not. פרדיקט זה הכרנו לראשונה בסעיף 7.2, כשדנו בחוקים המכילים קשרים לוגיים NOT ו-OR. בגירסת WISDOM-PROLOG הפרדיקט הזה מוגדר גם כאופרטור (ראה נספח ב׳). בהמשך נבטא את הפרדיקט not בצורה של אופרטור.

פרדיקט המערכת not	
בצורה של פרדיקט	בצורה של אופרטור
not(a)	not a
not(Goal)	not Goal
not(likes(tom, ann))	not likes(tom, ann)

איור 11.5 – ייצוג של not

את ההיכרות הראשונה עם הפרדיקט not נעשה באמצעות השאילתות הבאות:

```
?-  a.
no

?-  not a.
yes
```

בפרק הראשון הגדרנו את קשר השלילה NOT. אמרנו, כי ערך האמת של הפסוק NOT a מנוגד לזה של a. בסעיף זה נבחן איך פרדיקט המערכת not מיישם את ההגדרה הלוגית של הקשר NOT. נתאר גם את הסטיות מההגדרה הלוגית וננתח אותן.
העובדות והחוקים של תכנית לוגית הינם פסוקים בעלי ערך-אמת T(אמת). פירוש הדבר הוא, שתכנית לוגית מתארת עובדות וחוקים נכונים, ואיננה מתארת עובדות וחוקים שאינם נכונים.
את קנה המידה של תכנית לוגית לגבי נכונות או אי-נכונות של עובדה בסיסית ניתן לבטא כך:

- עובדה בסיסית, הנמצאת בתכנית, או
  הנובעת ממנה, היא עובדה נכונה
  (כלומר, ערך האמת שלה הינו T).

- עובדה בסיסית, אשר אינה נמצאת בתכנית וגם
  אינה נובעת ממנה – איננה נכונה
  (כלומר, ערך האמת שלה הינו F).

ניעזר בדוגמה: נציג שאילתות על בסיס התכנית The Family שבפרק 5 והתכנית Family Rules שבפרק 7. באיור 11.6 ריכזנו את השאילתות והתשובות, יחד עם החלקים הרלוונטיים של התכניות. העובדה הבסיסית female(leah) אינה נמצאת בתכנית וגם אינה נובעת ממנה. הוא הדין גם לגבי העובדה הבסיסית: father(abraham, ishmael).

```
parent(abraham,isaac).
parent(isaac,jacob).
parent(sara,isaac).
male(abraham).
male(isaac).
male(jacob).
female(sara).
father(X,Y) :- parent(X,Y), male(X).

| ?- not female(sara).

no
| ?- not female(leah).

yes

| ?- not father(abraham,ishmael).

yes

tracing: | ?- not father(abraham,ishmael).
[0] call: father(abraham,ishmael)?
 [1] call: parent(abraham,ishmael)?
 [1] fail: parent(abraham,ishmael)?
[0] fail: father(abraham,ishmael)?
... in 0 reductions.
```

איור 11.6 – שלילה של עובדה בסיסית

ה-trace של חישוב השאילתה not father(abraham, ishmael) נמצא באיור 11.6. ננתח את שלבי החישוב. כאשר מתקבלת השאילתה, מנסה מערכת הפרולוג להוכיח את נכונות העובדה father(abraham, ishmael). תהליך החישוב מסתיים בכישלון והמערכת מסיקה מכך כי שלילת העובדה אכן נכונה. כתוצאה משיקולים אלה מתקבלת תשובה חיובית.

.1	שאילתה	not father(abraham, ishmael)
.2	מטרה	father(abraham, ishmael)
.3	חוק	father(X, Y) :- parent(X, Y), male(X)
.4	הצבה	X = abraham, Y = ishmael
.5	מטרה	parent(abraham, ishmael)
.6	מסקנה	המטרה parent(abraham, ishmael) נכשלה ולכן המטרה father(abraham, ishmael) נכשלה ולכן המטרה not father(abraham, ishmael) הצליחה

איור 11.7 – שלבי החישוב של not father(abraham, ishmael)

תיארנו את שלבי החישוב באיור 11.7. מערכת הפרולוג מבצעת את תהליך החישוב על פי הכללים האלה:

- המטרה not Goal מצליחה על בסיס התכנית Program, אם המטרה Goal נכשלת על בסיס זה.
- המטרה not Goal נכשלת, אם המטרה Goal מצליחה.

הכללים האלה באים לידי ביטוי בתכנית הבאה:

```
not Goal :- Goal, !, fail.
not Goal.
```

תכנית 11.7 – שלילה באמצעות הכישלון

תכנית 11.7 הינה הגדרת השלילה, המשמשת גירסאות רבות של פרולוג. היא מכונה בשם **שלילה באמצעות הכישלון** (negation by failure). הפרדיקט fail הוא פרדיקט המערכת, אשר משמעותו כישלון.

התכנית פועלת בדרך זו: כאשר מתקבלת השאילתה not Goal, פונה המערכת אל החוק הראשון. אם המטרה Goal מצליחה, מצליח גם הפרדיקט Cut. בעקבותיו בא הפרדיקט fail, אשר משמעותו היא שהוכחת not Goal על פי החוק הראשון נכשלה. הפרדיקט Cut מונע את הנסיגה ופניה אל החוק השני, ולכן אם המטרה Goal מצליחה, המטרה not Goal נכשלת. להיפך, אם המטרה Goal איננה מצליחה, עוברת המערכת אל החוק השני ומבצעת את האיחוד. במקרה זה המטרה not Goal מצליחה.

התכנית 11.7 מכילה צירוף המכונה בשם **Cut-fail**. בצירוף זה משתמשים כדי לומר, שאם החישוב הצליח עד עתה – הגיע הזמן לנטוש אותו. "עד עתה" פירושו עד ה-Cut.

נעיין בכלל הבא:

- אם המטרה A מצליחה, אז המטרה P נכשלת, אחרת - המטרה P מצליחה.

את הכלל הזה ניתן לבטא כך:

```
P :- A, !, fail; true.
```

הפרדיקט true הינו פרדיקט המערכת שמשמעותו הצלחה.

את החוק האחרון ניתן "לפרק" לשני חוקים:

```
P :- A, !, fail.
P :- true.
```

גופו של החוק השני מורכב מהפרדיקט true. פירוש הדבר, שכאשר המטרה מתאחדת עם הראש P של החוק השני, המטרה מצליחה. במילים אחרות, החוק השני שקול לעובדה P.

את ה-Cut אין מבטאים בעת קריאת התכנית, ולכן תכנית לוגית עם Cut שקופה פחות מאשר תכנית ללא Cut. במקרה של המטרה P ניתן לעצב תכנית אלטרנטיבית, אשר בה נשתמש בפרדיקט not במקום הפרדיקט Cut. לשם כך ננסח את הכלל הקודם באופן הבא:

- המטרה P נכשלת אם המטרה A מצליחה.
- המטרה P מצליחה אם המטרה A נכשלת.

את הניסוח האחרון נתרגם לפרולוג כך:

```
P :- A, fail.
P :- not A, true.
```

בתכנית 11.8 הבאנו דוגמה של שימוש אלטרנטיבי ב-Cut וב-not. נציין כי בפרולוג העברי השלילה מתבטאת באמצעות פרדיקט המערכת "לא".

ים :- חופש, !, שמש. ים :- קיץ.	ים :- חופש, שמש. ים :- לא חופש, קיץ.

תכנית 11.8 – הפרוצדורה "ים" עם Cut ועם not

הפרולוג אינו מאפשר למשתמש להגדיר את פרדיקט המערכת. בנוסף, ה-trace איננו מפרט את שלבי חישוב המטרה, שהוגדרה באמצעות פרדיקט המערכת. כדי לפרט את ה-trace של הפרוצדורה not, הגדרנו את היחס nnot באופן זהה לחלוטין להגדרת היחס not. ובכך "רימינו" את המערכת וקיבלנו את ה-trace המפורט. את תוצאות "ההונאה" ניתן לראות באיור 11.8.

```
nnot(Goal) :-
   Goal,
   !,
   fail.
nnot(Goal).



! ?- nnot(father(abraham,isaac)).

no
! ?- trace.
tracing: ! ?- nnot(father(abraham,isaac)).
[0] call: nnot(father(abraham,isaac))?
 [1] call: father(abraham,isaac)?
  [2] call: parent(abraham,isaac)?
  [2] exit: parent(abraham,isaac)?
  [3] call: male(abraham)?
  [3] exit: male(abraham)?
 [1] exit: father(abraham,isaac)?
 [4] call: fail?
 [4] fail: fail?
[0] fail: nnot(father(abraham,isaac))?
```

איור 11.8 – ה- trace של not Goal

11.3 – המשתנה הלוגי ופרדיקט המערכת not

נעבור לשאילתות, שאינן בסיסיות, כלומר – לשאילתות אשר כוללות משתנה. נציג את השאילתה הבאה וננתח את התשובה.

```
?-  not father(abraham, X).
no
```

כמה מהשלבים לקבלת התשובה ניתנים באיור 11.9. בהתקבל השאילתה, המערכת של פרולוג פונה אל המטרה father(abraham, X). תהליך החישוב מסתיים בהצלחה ומתקבלת ההצבה X=isaac. מערכת הפרולוג מסיקה מכך כי שלילת המטרה נכשלה, ולכן אנו מקבלים תשובה שלילית.

.1	שאילתה	not father(abraham, X)
.2	מטרה	father(abraham, X)
.3	חוק	father(abraham, X) :- parent(abraham, X), male(abraham).
.4	מטרה	parent(abraham, X)
.5	הצבה	X = isaac
.6	מסקנה	המטרה parent(abraham, isaac) הצליחה המטרה male(abraham) הצליחה המטרה father(abraham, isaac) הצליחה ולכן תהליך החישוב של father(abraham, X) הסתיים בהצלחה ולכן תהליך החישוב של not father(abraham, X) הסתיים בכישלון

איור 11.9 – שלבי החישוב של not father(abraham, X)

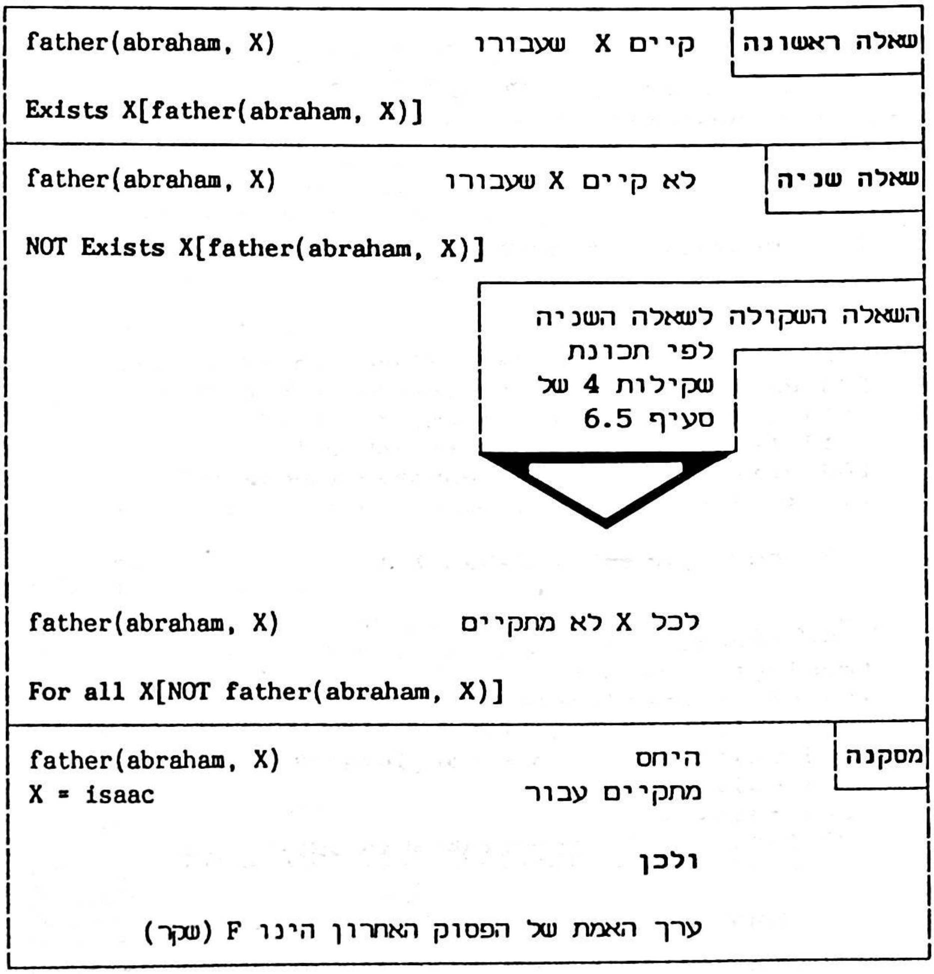
את קבלת התשובה השלילית ניתן לראות מזווית ראיה נוספת. הצגת השאילתה father(abraham, X) כמוה כהצגת השאלה הבאה:

האם קיים X שעבורו מתקיים היחס ?-father(abraham, X)

כפי שראינו, התשובה לשאלה הזאת הינה חיובית.
אם-כן, הצגת השאילתה not father(abraham, X) מתפרשת כך:

האם זה נכון שלא קיים X, אשר מקיים את היחס
? father(abraham, X)

לשאלה זו נענינו בשלילה, שהרי ידוע כי ההצבה של isaac תחת X מקיימת את היחס .father(abraham, X)
את שתי השאלות האחרונות ניתן לתאר בצורה פורמלית. לשם כך ניעזר במינוח של לוגיקת היחסים. איור 11.10 מציג את התאור הפורמלי. באיור זה ביטאנו את הכמת הקיומי באמצעות המילה **Exists** (קיים), ואת הכמת הכללי באמצעות הצירוף **For all** (לכל).



איור 11.10 – השאלות במינוח של לוגיקת היחסים

בעת החישוב של ,father(abraham, X)
חישבנו גם את המטרה .parent(abraham, X)

הבה נתרכז בחלק זה של החישוב ונעיין בשאילתות ובתשובות הבאות:

```
?- not  parent(abraham, ishmael).
yes
?- not  parent(abraham, jacob).
yes
?- not  parent(abraham, X).
no
```

ישנם עצמים רבים, אשר על פי התכנית The Family, אברהם אינו האב שלהם: ישמעאל, יעקב, ועוד. אף-על-פי-כן, התשובה לשאילתה האחרונה הינה שלילית. כלומר, אין זה מספיק להוכיח כי קיים עצם X כזה, שאברהם אינו אביו. כפי שהדגמנו באיור 11.10, המשתנה X של השאילתה האחרונה אינו משתנה המכומת קיומי, אלא הוא מכומת כללי. על-כן, קבלת תשובה חיובית מותנית בכך, שעל פי התכנית The Family, אברהם אינו אביו של אף אחד, אבל הוא אביו של יצחק ולכן קיבלנו תשובה שלילית.

במינוח הפרוצדורלי ניתן לומר, כי המטרה not parent(abraham, X) נכשלה משום שהמטרה parent(abraham, X) הצליחה בשל האיחוד בינה לבין parent(abraham, isaac).
את ה-trace המפורט של not parent(abraham, X) ניתן לראות באיור 11.11. לשם ההשוואה הבאנו גם את ה-trace של not parent(abraham, ishmael).

```
| ?- nnot(parent(abraham,ishmael)).

yes
| ?- trace.
tracing: | ?- nnot(parent(abraham,ishmael)).
[0] call: nnot(parent(abraham,ishmael))?
 [1] call: parent(abraham,ishmael)?
 [1] fail: parent(abraham,ishmael)?
[0] exit: nnot(parent(abraham,ishmael))?
... in 1 reductions.

| ?- nnot(parent(abraham,X)).

no
| ?- trace.
tracing: | ?- nnot(parent(abraham,X)).
[0] call: nnot(parent(abraham,X81))?
 [1] call: parent(abraham,X81)?
 [1] exit: parent(abraham,isaac)?
 [2] call: fail?
 [2] fail: fail?
[0] fail: nnot(parent(abraham,_81))?
```

איור 11.11 – המטרות not Goal: עם ובלי המשתנה

עתה נציג שאילתה מורכבת, אשר אחד מביטוייה מכיל את הפרדיקט not, והאחר אינו מכיל אותו. המשתנה X בביטוי male(X) מכומת קיומי. אותו משתנה ב-not parent(abraham, X) מכומת כללי. נבדוק איך מערכת הפרולוג מטפלת במצב הזה. ניעזר לשם כך באיור 11.12.

```
| ?- male(X), not parent(abraham,X).

X = abraham ;

X = jacob ;

no


| ?- trace.
tracing: | ?- male(X), not parent(abraham,X).
[0] call: male(X80)?
[0] exit: male(abraham)?
[1] call: parent(abraham,abraham)?
[1] fail: parent(abraham,abraham)?
... in 1 reductions.

X = abraham ;
[0] redo: male(abraham)?
[0] exit: male(isaac)?
[1] call: parent(abraham,isaac)?
[1] exit: parent(abraham,isaac)?
[0] redo: male(isaac)?
[0] exit: male(jacob)?
[1] call: parent(abraham,jacob)?
[1] fail: parent(abraham,jacob)?
... in 1 reductions.

X = jacob ;
[0] redo: male(jacob)?
[0] fail: male(_80)?

no

| ?- not parent(abraham,X), male(X).

no
| ?- trace.
tracing: | ?- not parent(abraham,X), male(X).
[0] call: parent(abraham,X32)?
[0] exit: parent(abraham,isaac)?


no
```

איור 11.12 – המופע של not בסדרה של מטרות

התשובה לשאילתה אשר בה הפסוק male(X) הינו במקום הראשון, נובעת באופן לוגי מתכנית The Family. לתשובה זו היינו מצפים לו היינו שואלים מי הם הגברים של The Family, אשר

אברהם אינו האב שלהם. בשאילתה השניה שינינו את סדר הפסוקים ועקב כך גרמנו לשינוי במשמעות השאלה. דבר זה מבטא את הגישה הפרוצדורלית, שעל פיה הגדרנו את הפרדיקט not. סדר החוקים בתכנית 11.5 משפיע על תוצאות החישוב בגלל הופעתו של Cut.
מכך אנו לומדים, שכאשר הפרדיקט not מופיע בתוך סדרת המטרות הקשורות באמצעות AND, התשובות תלויות בסדרן של המטרות. נעיין בתכנית המגדירה את היחס unhappy; ונבדוק את עקביות התשובות, שהתקבלו על בסיס התכנית הזאת (ראה איור 11.13). התשובות:

```
unhappy(X) :- not healthy(X), young(X).
healthy(tom).
young(john).

?- unhappy(john).
yes
?- unhappy(X).
no
```

איור 11.13 – תשובות שאינן עקביות

המטרה unhappy(john) הצליחה, משום שראשו של החוק המגדיר את היחס unhappy התאחד עם unhappy(john). אחרי ביצוע ההצבה X=john בגופו של החוק, פנתה מערכת הפרולוג אל המטרה not healthy(john). המטרה הזאת הצליחה וגם המטרה young(john) הצליחה ולפיכך התקבלה תשובה חיובית.
לעומת זאת המטרה unhappy(X) נכשלה, משום שהמטרה healthy(X) הצליחה. יש לבחון איפוא את התכניות של פרולוג המכילות את הפרדיקט not, בתשומת לב רבה. העצה המעשית הינה לרשום את הפסוק שמכיל not בקצה גופו של החוק, כדי להגיע למטרה not Goal אחרי ביצוע ההצבות. נגדיר למשל, את היחס unhappy1 באמצעות החוק הבא:

```
unhappy1(X) :- young(X), not healthy(X).
?- unhappy1(X).
X = john.
```

על אף הקשיים שנגרמים בשל השימוש בפרדיקט not, לא ניתן להתעלם ממנו. באמצעותו למשל, ניתן להגדיר שתי רשימות זרות:

```
disjoint(List1, List2) :-
                 not (member(X, List1), member(X, List2))
```

נציין כי מבחינה לוגית הפסוק:

```
NOT (member(X, List1) AND member(X, List2))
```

שקול לפסוק:

NOT member(X, List1) OR NOT member(X, List2)

את תכונות השקילות של הפסוקים למדנו בפרקים 2 ו-6. איור 11.14 מראה, כי מערכת הפרולוג אינה מכירה את הפסוקים לעיל כשקולים מבחינה סמנטית. דבר זה מזכיר שוב, כי ההגדרה הלוגית של NOT והגדרת הפרדיקט not תואמות זו את זו עד גבול מסויים. בנוגע לאיור 11.14 נציין, כי היחס member מוגדר ב-WISDOM PROLOG בתור פרדיקט המערכת ולכן נאלצנו "לרמות" את פרולוג ולהגדיר את היחס mmember באופן זה לזה של member.

```
disjoint1(List1, List2) :-
        not (mmember(X, List1), mmember(X, List2)).

disjoint2(List1, List2) :-
        not mmember(X, List1); not mmember(X, List2)).

mmember(X, [X|Tail]).
mmember(X, [Y|Tail]) :- mmember(X,Tail).


| ?- disjoint1([a,b,c],[d,e,f]).

yes

| ?- trace.
tracing: | ?- disjoint1([a,b,c],[d,e,f]).
[0] call: disjoint1([a,b,c],[d,e,f])?
 [1] call: mmember(X183,[a,b,c])?
 [1] exit: mmember(a,[a,b,c])?
 [2] call: mmember(a,[d,e,f])?
  [3] call: mmember(a,[e,f])?
   [4] call: mmember(a,[f])?
    [5] call: mmember(a,[])?
    [5] fail: mmember(a,[])?
   [4] fail: mmember(a,[f])?
  [3] fail: mmember(a,[e,f])?
 [2] fail: mmember(a,[d,e,f])?
 [1] redo: mmember(a,[a,b,c])?
  [2] call: mmember(X277,[b,c])?
  [2] exit: mmember(b,[b,c])?
 [1] exit: mmember(b,[a,b,c])?
 [3] call: mmember(b,[d,e,f])?
  [4] call: mmember(b,[e,f])?
   [5] call: mmember(b,[f])?
    [6] call: mmember(b,[])?
    [6] fail: mmember(b,[])?
   [5] fail: mmember(b,[f])?
  [4] fail: mmember(b,[e,f])?
 [3] fail: mmember(b,[d,e,f])?
 [1] redo: mmember(b,[a,b,c])?
  [2] redo: mmember(b,[b,c])?
   [3] call: mmember(X361,[c])?
   [3] exit: mmember(c,[c])?
  [2] exit: mmember(c,[b,c])?
 [1] exit: mmember(c,[a,b,c])?
```

```
[4] call: mmember(c,[d,e,f])?
 [5] call: mmember(c,[e,f])?
  [6] call: mmember(c,[f])?
   [7] call: mmember(c,[])?
   [7] fail: mmember(c,[])?
  [6] fail: mmember(c,[f])?
 [5] fail: mmember(c,[e,f])?
[4] fail: mmember(c,[d,e,f])?
[1] redo: mmember(c,[a,b,c])?
 [2] redo: mmember(c,[b,c])?
  [3] redo: mmember(c,[c])?
   [4] call: mmember(X445,[])?
   [4] fail: mmember(X445,[])?
  [3] fail: mmember(_361,[c])?
 [2] fail: mmember(_277,[b,c])?
[1] fail: mmember(_183,[a,b,c])?
[0] exit: disjoint1([a,b,c],[d,e,f])?
... in 1 reductions.


! ?- disjoint2([a,b,c],[d,e,f]).

no


! ?- trace.
tracing: ! ?- disjoint2([a,b,c],[d,e,f]).
[0] call: disjoint2([a,b,c],[d,e,f])?
 [1] call: mmember(X183,[a,b,c])?
 [1] exit: mmember(a,[a,b,c])?
 [1] call: mmember(_183,[d,e,f])?
 [1] exit: mmember(d,[d,e,f])?
 [1] call:  not mmember(_183,[a,b,c]) ;  not mmember(_183,[d,e,f])?
 [1] fail:  not mmember(_183,[a,b,c]) ;  not mmember(_183,[d,e,f])?
[0] fail: disjoint2([a,b,c],[d,e,f])?
```

איור 11.14 – הפרוצדורות disjoint1 ו-disjoint2 אינן שקולות

11.4 השלילה וה-Cut – תרגול

בסעיף הזה ניתנים מספר תרגילים בנושאי השלילה וה-Cut. התרכזנו בשלושת הנושאים האלה:

– השימוש ב-Cut.
– הקשר בין הפרדיקטים not ו-Cut והשימוש האלטרנטיבי בהם.
– ההבדל בין הקשר הלוגי NOT לבין פרדיקט המערכת not.

שאלה 1

א. נתון החוק הבא:

```
a :- not b.
```

מצא את התשובה לשאילתה הבאה על פי החוק הנתון:

```
?- a.
```

ב. נתון החוק הבא:

```
a :-  not(not b).
```

מצא על פי החוק הזה את התשובה לשאילתה הבאה:

```
?- a.
```

שאלה 2

נתונים שני החוקים הבאים:

```
a :- not b.
b :- not a.
```

א. האם החוקים האלה סותרים זה את זה? נמק את תשובתך.

הערה. במינוח של פרק 3 ניתן לנסח את השאלה בדרך זו: האם הקבוצה שמורכבת משני חוקים אלה, הינה עקבית?

ב. בדוק מהי תשובת הפרולוג לשאילתות הבאות:

```
?- a.
?- b.
```

שאלה 3

נתונים שני החוקים הבאים:

```
a :- not b.
b :- a.
```

א. האם החוקים האלה סותרים זה את זה? נמק את תשובתך.

ב. בדוק מהי תשובת הפרולוג לשאילתות הבאות:

```
?- a.
?- b.
```

שאלה 4

בפרק 2 למדנו כי הפסוק NOT(a AND b) שקול מבחינה סמנטית לפסוק NOT a OR NOT b.
להלן ההגדרות של היחסים p1, p2 ו-p3:

```
p1 :- not (a,b).
p2 :- not a; not b.
p3 :- not a.
p3 :- not b.
```

האפשרויות לגבי העובדות a ו-b הן כדלהלן:

א. שתי העובדות a ו-b נמצאות בתכנית.

ב. אחת מהעובדות a או b נמצאת בתכנית.

ג. אף אחת מהעובדות אינה נמצאת בתכנית.

בדוק בכל אחד מהאפשרויות א', ב', ג', מהן התשובות לשאילתות הבאות:

```
?- p1.
?- p2.
?- p3.
```

שאלה 5

נתון הכלל הבא:

p IF a AND b OR NOT a AND c

תרגם אותו לפרולוג:

א. באמצעות הפרדיקט Cut.

ב. באמצעות הפרדיקט not.

הערה: את הקשרים הלוגיים שמופיעים בכלל, יש לפרש בהתאם לסדר קדימויות של הקשרים (ראה פרק 1).

שאלה 6

ביטאנו את המקסימום של שני המספרים באמצעות הכלל הבא:
אם X גדול מ-Y, אז המקסימום הוא X, אחרת – המקסימום הוא Y.
רצינו לבטא את הכלל הזה באמצעות היחס max בין שלושת הארגומנטים, כאשר הארגומנט השלישי הינו המקסימום של שני הארגומנטים הראשונים. לשם כך כתבנו את החוקים האלה:

```
max(X, Y, X) :- X > Y, !.
max(X, Y, Y).
```

הצגנו שאילתה וקיבלנו תשובה:

```
?- max(5, 2, 2).
yes
```

לפי התשובה הזאת, 2 הוא המקסימום של 5 ו-2 וכידוע – תשובה זו אינה נכונה.

א. הסבר מדוע קיבלנו תשובה זו.

ב. האם החוקים האלה מבטאים את הכלל? נמק את תשובתך.

ג. הגדר יחס max1 כך, שעל פיו נקבל תשובה שלילית לשאילתה:
max1(3, 2, 2) .

שאלה 7

לפניך שתי גירסאות נוספות של החוק הרקורסיבי אב_בשושלת.

```
א. אב_בשושלת1(X, Y) :- אב(X, Y), !.
   אב_בשושלת1(X, Y) :- אב(X, L), אב_בשושלת1(L, Y).

ב. אב_בשושלת2(X, Y) :- אב(X, Y).
   אב_בשושלת2(X, Y) :- אב(X, L), !, אב_בשושלת2(L, Y).
```

נתח והשווה את שתי הגירסאות של אב_בשושלת.

שאלה 8

הצגנו שאילתה מורכבת על פי התכנית The Family וקיבלנו את התשובות הבאות:

```
?- male(X), not parent(X).
X = abraham;
X = isaac;
X = jacob;
```

א. תן הסבר לתשובות שהתקבלו.
ב. מצא את התשובות לשאילתה הבאה:

?-not parent(X), male(X).

שאלה 9

הגדרנו את היחס disjoint3. הצגנו שאילתה ועקבנו אחר תהליך קבלת התשובה:

```
| disjoint3(List1, List2) :- not member(X, List1).
| disjoint3(List1, List2) :- not member(X, List2).
| ?- disjoint3([a,b,c],[d,e,f]).
no
| ?- trace.
tracing: ?- disjoint3([a,b,c], [d,e,f]).
[0] call: ?- disjoint3([a,b,c], [d,e,f])?
 [1] call: member(X182, [a,b,c])?
 [1] exit: member(a, [a,b,c])?
 [1] call: member(X182, [d,e,f])?
 [1] exit: member(d, [d,e,f])?
[0] fail: disjoint3([a,b,c], [d,e,f])?
no
```

א. השווה בין היחסים disjoint2 ו-disjoint3. האם ניתן לומר כי הם מבטאים את אותו כלל?
ב. חזור את שאלה 4 והשווה:
בין disjoint1 לבין p1.
בין disjoint2 לבין p2.
בין disjoint3 לבין p3.
ג. הסבר, מדוע התשובות לשאילתות שבשאלה 4 זהות זו לזו ולעומת זאת, התשובות לשאילתות הבאות אינן זהות:

```
?- disjoint1([a,b,c], [d,e,f]).
?- disjoint2([a,b,c], [d,e,f]).
```

שאלה 10

נעיין ביחס disjoint4, המוגדר להלן:

```
disjoint4(List1, List2) :- member(X, List1),
                           not member(X, List2).
```

א. מה מבטא היחס הזה?

ב. האם disjoint4 שקול לאחד מהיחסים disjoint1, disjoint2, disjoint3?
את המונח "שקול" בהקשר זה נבין כך: שני יחסים שקולים מבטאים אותו כלל ונותנים תשובות זהות לשאילתות זהות.
נמק את תשובתך.

שאלה 11

א. מצא את התשובות לשאילתה הבאה:

```
?- not not member(X, List).
```

ב. מצא את התשובות לשאילתות הבאות:

```
?- not disjoint1(List1, List2).
?- not disjoint4(List1, List2).
```

שאלה 12

א. באיור 11.15 הוצגו ההגדרות של היחסים unhappy2, unhappy3, unhappy4. נתח אותן והשווה בינן לבין unhappy1. האם יש מבין הגירסאות החדשות לפחות אחת השקולה סמנטית ליחס unhappy1? נמק את תשובתך.

ב. מצא את התשובות לשאילתות הבאות:

```
?- not unhappyN(X), N=1, 2, 3, 4
?- not not unhappyN(X), N=1,2,3,4
```

הערה: ניתן להיעזר ב-trace, שבאיור 11.15.

```
unhappy2(X) :- young(X), healthy(X), !, fail.
unhappy2(X) :- young(X).
```

```
| ?- unhappy2(X).

X = john
```

```
| ?- trace.
tracing: | ?- unhappy2(X).
[0] call: unhappy2(X79)?
 [1] call: young(X169)?
 [1] exit: young(john)?
 [2] call: healthy(john)?
 [2] fail: healthy(john)?
 [1] redo: young(john)?
 [1] fail: young(_169)?
 [1] call: young(X169)?
 [1] exit: young(john)?
[0] exit: unhappy2(john)?
... in 2 reductions.

X = john
```

```
| unhappy3(X) :- healthy(X), !, fail.
| unhappy3(X) :- young(X).

| ?- unhappy3(X).

no
| ?- trace.
tracing: | ?- unhappy3(X).
[0] call: unhappy3(X79)?
 [1] call: healthy(X169)?
 [1] exit: healthy(tom)?
 [2] call: fail?
 [2] fail: fail?
[0] fail: unhappy3(_79)?
```

```
| unhappy4(X) :- healthy(X), fail.
| unhappy4(X) :- young(X).

| ?- unhappy4(X).

X = john
```

```
| ?- trace.
tracing: | ?- unhappy4(X).
[0] call: unhappy4(X79)?
 [1] call: healthy(X169)?
 [1] exit: healthy(tom)?
 [2] call: fail?
 [2] fail: fail?
 [1] redo: healthy(tom)?
 [1] fail: healthy(_169)?
 [1] call: young(X169)?
 [1] exit: young(john)?
[0] exit: unhappy4(john)?
... in 2 reductions.

X = john
```

איור 11.15 – גירסאות שונות של הפרוצדורה unhappy

פרק 12

פרדיקטים אקסטרה-לוגיים ומטה-לוגיים

בפרק זה נכיר פרדיקטי מערכת נוספים. נציין, כי התכניות שבפרק זה הורצו בגירסת WISDOM PROLOG. לקורא, אשר גירסה אחרת של פרולוג נמצאת ברשותו מומלץ לעיין במדריך המצורף לגירסתו ולתקן את התכניות בהתאם, לפני הרצתן.
נציג את יעדי הפרק.

- בסעיף 12.1 נכיר פרדיקטים חשובים. נסווג אותם בהתאם לתפקידים שהם ממלאים במערכת הפרולוג.

קריאה וכתיבה של ביטוים	–	read, write
גישה לתווים	–	get, put, name
עריכה של פלט	–	tab, nl
בדיקה של סוגי נתונים	–	atom, integer, var
גישה לביטוים מורכבים	–	functor, arg
פניה לפיסקאות	–	clause, assert, listing
בקרת זרימה	–	call, exit, fail, repeat
קריאת קבצים	–	consult, reconsult

- בסעיף 12.2 נלמד להשתמש בפרדיקטי המערכת. נדגים את תכונותיהם באמצעות תכניות ואיורים.
- בסעיף 12.3 נסווג את פרדיקטי המערכת על פי השתייכותם לקבוצות אלו:
  אקסטרה-לוגיים – פרדיקטים אשר התנהגותם חורגת ממסגרת של תכנות לוגי.
  מטה-לוגיים – פרדיקטים שמטפלים בפיסקאות של תכנית לוגית ומאפשרים לבצע תכנות-על.
- בסעיף 12.4 נביא שאלות לעבודה עצמית בנושא פרדיקטי המערכת.

12.1 סקירה של פרדיקטי המערכת

בסעיף זה נציג את פרדיקטי המערכת הנפוצים ביותר. נציין כי אופן פעולתם של רבים מהם תלוי באמצעי רקע, כגון: חומרה, מערכת ההפעלה וכו'. במקרים רבים, פעולתם חורגת ממסגרת של תכנות לוגי והיא יכולה להיות שונה בגירסאות שונות של פרולוג. בעת קריאת סעיף זה מומלץ להפעיל את הפרדיקטים המוזכרים, כדי להכיר את אופן פעולתם בגירסה הנמצאת ברשות הקורא. אין לצפות לשליטה מוחלטת בפרדיקטים אלה מתוך היכרות ראשונית. השליטה בהם באה לאחר יישום ועבודה ממושכים.

קריאה וכתיבה של ביטויים

- **read(Term)** – באמצעות הפרדיקט הזה קוראים ביטוי מלוח המקשים או מתוך הקובץ הנתון. כאשר Term הינו משתנה, מתבצעת ההצבה של ביטוי תחת Term. במקרה של כישלון אין נסיגה.

- **write(Term)** – באמצעות הפרדיקט הזה כותבים ביטוי על המסך או אל תוך הקובץ הנתון. במקרה של כישלון אין נסיגה.

התכנית שלהלן מתארת את אופן פעולתם של הפרדיקטים read ו-write היא קוראת את המספר N באמצעות read, מחשבת את ריבועו R, מציגה את התשובה R על המסך באמצעות write ומסתיימת לאחר קריאת stop. גירסה נוספת של התכנית הזאת מופיעה בסעיף הבא (ראה איור 12.4).

```
ribua :- read(N),
         proceed(N).
proceed(stop) :- !.
proceed(N) :- R is N*N,
              write(R),
              ribua.
```

תכנית 12.1 – ריבוע

גישה לתווים

- **get0(Character)** – הפרדיקט משמש לקריאה של תו והצבת קוד ASCII שלו תחת המשתנה Character.

- **get(Character)** – הפרדיקט משמש לקריאה של תו, ואם ניתן להדפיסו – להצבת קוד ASCII שלו תחת המשתנה Character. הפרדיקט מדלג על תווים אשר לא ניתן להדפיסם.

- **put(Character)** – הפרדיקט משמש לכתיבה של תו, אשר קוד ASCII שלו הוצב תחת המשתנה Character.

התכנית שלהלן מתארת את אופן פעולתם של הפרדיקטים get ו-put. היא קוראת באמצעות הפרדיקט get את התווים הניתנים להדפסה, ומחזירה אותם באמצעות הפרדיקט put. גם אל התכנית הזאת נחזור בסעיף הבא.

```
go :- get(C),
      put(C),
      go_on(C).
go_on(46) :- !.
go_on(C) :- go.
```

תכנית 12.2 – קריאה וכתיבה של תווים

- **name(Atom,List_of_Codes)** – הפרדיקט הזה מקשר בין האטום Atom לבין רשימת קודים ASCII של תווים המרכיבים אותו. הפרוצדורה שלהלן מגדירה קבוצות זרות של תווים ומדגימה את השימוש בפרדיקט name. בפרוצדורה זו נדון בהרחבה בסעיף הבא.

```
disjoint_characters(Atom1, Atom2) :-
        name(Atom1, List1), name(Atom2, List2),
        not (member(X, List1), member(X, List2)).
```

ראה סעיף 9.2 – member

תכנית 12.3 – קבוצות זרות של תווים

עריכה של הפלט

- **tab(N)** – הפרדיקט משמש לקביעת N רווחים. ניתן להגדיר אותו כך:

```
tab(0) :- !.
tab(N) :- N>0, put(32),
          N1 is N-1, tab(N1).
```

תכנית 12.4 – הגדרה של הפרדיקט tab

- **nl** – הפרדיקט משמש לקביעת שורה חדשה. האותיות nl הינן ראשי התיבות של המילים new line.

את פעולת הפרדיקטים tab ו-nl נדגים באמצעות שאילתה ותשובה:

```
?- put(65), nl, tab(1), put(66), nl, tab(2), put(67).
A
 B
  C
```

דוגמה נוספת של עריכת הפלט נציג בסעיף הבא, כשנדון בפרוצדורה ribua.

בדיקה של סוגי הנתונים

- **atom(X)** – המטרה atom(X) מצליחה, אם X הינו אטום. במקרים בהם X איננו אטום, פעולת הפרדיקט atom תלויה בגירסה של פרולוג.

- **integer(X)** – המטרה הזאת מצליחה, אם X הינו מספר. במקרים בהם X איננו מספר, פעולתו של הפרדיקט תלויה בגירסה.

- **atomic(X)** – המטרה מצליחה, כאשר X הינו אטום או מספר. את היחס atomic ניתן להגדיר כך:

```
atomic(X) :- atom(X).
atomic(X) :- integer(X).
```

תכנית 12.5 – הגדרה של הפרדיקט atomic

- **var(X)** – המטרה מצליחה, כאשר X הינו משתנה.

- **nonvar(X)** – המטרה מצליחה, כאשר X אינו משתנה.

איור 12.1 מדגים בדיקה של סוגי נתונים, שנעשתה בגירסה 2.10 של WISDOM-PROLOG.

האטום a	המשתנה A	המספר 5
?- atom(a). yes	?- atom(A) no	?- atom(5). no
?- integer(a). no	?- integer(A). no	?- integer(5). yes
?- atomic(a). yes	?- atomic(A). no	?- atomic(5). yes
?- var(a). no	?- var(A). A=A27	?- var(5). no
?- nonvar(a). yes	?- nonvar(A). no	?- nonvar(5). yes

איור 12.1 – בדיקת סוגי הנתונים

תכנית 12.6 משתמשת בפרדיקטים integer ו-var לבדיקת סוגי הנתונים. השימוש באופרטור is נעשה בהתאם לתוצאות הבדיקה.

```
plus(X,Y,Z) :- integer(X), integer(Y), var(Z),
               Z is X+Y.
plus(X,Y,Z) :- integer(Y), integer(Z), var(X),
               X is Z-Y.
plus(X,Y,Z) :- integer(X), integer(Z), var(Y),
               Y is Z-X.
```

תכנית 12.6 – בדיקה של סוג הנתונים

גישה לביטויים מורכבים -

- **functor(Term, Functor, Arity)** – הביטוי Term הינו בעל הפנקטור הראשי Functor של אריטי Arity.

```
?- functor(parent(abraham, isaac), F, A).
F = parent
A = 2
```

- **arg(N, Term, Argument)** – הארגומנט ה-N-י של הביטוי Term הינו Argument.

```
?- arg(2, parent(abraham, isaac), Arg).
Arg = isaac
```

בסעיף 12.4 נציג את התכנית Subterm, בה נדגים את השימוש בפרדיקטים functor ו-arg.

פניה לפיסקאות

- **clause(Head, Body)** – הפרדיקט הזה מאתר פיסקה בעלת הראש הנתון Head.

```
?- clause(parent(abraham, isaac), Body).
Body = true
```

באיור 12.9 נביא דוגמאות נוספות של שימוש בפרדיקט clause.

- **assert(Clause)** – באמצעות הפרדיקט הזה מוסיפים פיסקה אל סוף הפרוצדורה. את השימוש בו נדגים באיור 12.10.
- **assertz(Clause)** – פועל כמו assert.
- **asserta(Clause)** – באמצעות הפרדיקט הזה מוסיפים פיסקה אל התחלת הפרוצדורה. את השימוש בו נדגים באיור 12.10.

- **retract(Clause)** – באמצעות הפרדיקט הזה מבטלים פיסקה. בסעיף הבא נראה איך משתמשים בו.

  נציין, כי אם הפיסקה Clause הינה חוק, יש להוסיף זוג נוסף של סוגריים בשל סיבות טכניות הקשורות לסדר הקדימויות של האופרטורים.

- **abolish(F, A)** – מבטל את כל הפיסקאות בעלות פנקטור F של אריטי A. נדגים את השימוש בו באיור 12.10.

- **listing** – באמצעותו של הפרדיקט הזה מערכת הפרולוג מפרטת את כל הפיסקאות שהוגדרו על ידי המשתמש.

- **listing(Procedure)** – באמצעותו המערכת מפרטת את הפיסקאות של הפרוצדורה Procedure. האיור 12.10 מדגים את השימוש בפרדיקט הזה.

בקרת הזרימה

- **call(X)** – הביטוי, אשר הוצב תחת X, מתקבל כמטרה. את הפרדיקט הזה היכרנו בעת דיון בפעולת ה-trace בפרק 7.

- **exit** – בגירסת WISDOM-PROLOG זהו סיום הרצה של מערכת הפרולוג והחזרת המשתמש אל מערכת הפעלה.

- **fail** – נכשל תמיד, ובכך גורם להמשך החיפוש או להודעת הכישלון.

- **true** – מצליח תמיד.

- **repeat** – מצליח מספר פעמים כרצוננו. ניתן לראותו כמוגדר באופן הבא:

```
repeat.
repeat :- repeat.
```

תכנית 12.7 – הגדרה של הפרדיקט repeat

קריאת קבצים

את אוסף העובדות והחוקים של תכנית הפרולוג כותבים בדרך כלל ב-text-editor. את הטענת הקובץ אל מערכת הפרולוג מבצעים באמצעות הפרדיקטים הבאים:

- **consult(File)** – קורא קובץ File. בעת הקריאה מארגן את הפיסקאות על פי השתייכותן לפרוצדורה, ובתוך הפרוצדורה הנתונה מארגן את הפיסקאות על פי סדר כתיבתן (בנושא זה ראה תכנית "התפתחות של בעלי-חיים" ואיור 30 בחטיבה רביעית).

- **[+File]** – סימון תחבירי נוסף של consult.

- **[+File1, +File2, +File3]** – ניתן לקרוא קבצים אחדים בזה אחר זה.

- **reconsult(File)** – קורא קובץ File. בעת קריאת הפרוצדורה מוחק פיסקאות הנמצאות במערכת הפרולוג והשייכות לפרוצדורה הזאת.

- **[File]** – סימון תחבירי נוסף של reconsult.

- **[File1, File2, File3]** – ניתן לקרוא באמצעות reconsult קבצים אחדים בזה אחר זה.

שם הקובץ הינו אטום. נציין גם, כי ניתן לקרוא כמה קבצים בזה אחר זה, כאשר אחדים מהם קוראים באמצעות consult ואחרים – באמצעות reconsult.

נדגים את ההבדל שבין consult לבין reconsult. נניח, כי נתונים שני קבצים: קובץ parent1, המורכב מפיסקה parent(tom, dick) וקובץ parent2, המורכב מפיסקה parent(liz, dick). תחילה נטען את הקובץ parent1. אם נטען לאחר מכן את הקובץ parent2 באמצעות consult- יהיו במערכת הפרולוג שתי פיסקאות של הפרוצדורה parent. אם נטען את הקובץ parent2 באמצעות reconsult- הפיסקה parent(tom, dick) תמחק, ובמערכת הפרולוג תהיה פיסקה אחת בלבד: parent(liz, dick).

הפרדיקטים שלעיל הוצגו על-פי הגירסה WISDOM-PROLOG. נציין שבגירסות אחרות יתכן וישנם שינויים בהגדרתם ובתיפקודם.

12.2 שימוש בפרדיקטים של המערכת

בסעיף זה נביא דוגמאות שונות המתארות את אופן פעולתם של פרדיקטי המערכת.

קריאה וכתיבה של ביטוים, עריכה של פלט (איורים 12.2, 12.3, 12.4)

את פעולתו של הפרדיקט write(Term) נדגים באמצעות איור 12.2. בחרנו להציב תחת המשתנה Term ארבעה סוגי ביטויים: ביטוי מורכב, משתנה, ביטוי שאינו בנוי בהתאם לכללי התחביר של פרולוג ואטום. בתור אטום שימש הביטוי הקודם, כשהוא מוקף בגרש מכל צעד. כמו כן הצגנו שתי שאילתות מורכבות.

לפרדיקט write(Term) יש שימוש נוסף, שגם אותו היצגנו באיור 12.2. אם נציב תחת Term ביטוי המוקף בגרשיים, נקבל בתשובה רשימת קודים ב-ASCII, אשר מייצגת אותו. הפרדיקט name(Sentence, List) מאשר, שרשימת הקודים אותה קיבלנו, אכן מייצגת את הביטוי שמוקף בגרשיים.

```
A>wisdom
Wisdom(tm) Prolog interpreter version 2.10
Copyright (C) M. Safra & M.Goldberg(MAXBYTE), Israel
Copyright (C) MIT Press (U.S.A)

yes
| ?- write(likes(tom,sisters(ann,liz))).
likes(tom,sisters(ann,liz))
yes
| ?- write(X).
X27
X = X27

yes
| ?- write(tom likes ann and liz).
Syntax error :

Here :
write( tom likes ann and liz )


| ?- write('tom likes ann and liz').
tom likes ann and liz
yes
| ?- write(X), X = likes(tom,ann).
X28
X = likes(tom,ann)

yes
| ?- X =likes(tom,ann), write(X).
likes(tom,ann)
X = likes(tom,ann)

yes
| ?- write("tom likes ann").
[116,111,109,32,108,105,107,101,115,32,97,110,110]
yes
| ?- name(Sentence,[116,111,109,32,108,105,107,101,115,32,97,110,110]).

Sentence = tom likes ann

yes
| ?-
|
```

איור 12.2 – השימוש ב-write

נבחן את פעולתו של הפרדיקט read(Term). לשם כך נעיין בפרוצדורה wrong_ribua, המוצגת באיור 12.3. כשמציגים את השאילתה ?-wrong_ribua, מופיע הסימן "|:" המעיד, שמערכת הפרולוג מחכה לקלט. כשמקישים את המספר 7, המטרה read(stop) נכשלת, והמערכת נסוגה אל החוק השני. שוב מופיע הסימן "|:". הפעם המשתנה N מתאחד עם המספר המוקש 913 ואנחנו זוכים לתשובה 833569, אשר היא הריבוע של 913. אם-כן, לא זכינו לדעת מהו הריבוע של 7.

את השאלה, מדוע אחרי הקשת stop מופיעה הודעת השגיאה, נציג באחת השאלות של סעיף 12.4.

```
wrong_ribua :- read(stop) , !.
wrong_ribua :- read(N),  R  is  N * N ,
               write(R) , wrong_ribua.


yes
| ?- wrong_ribua.
| 7.
| 913.
833569.

yes
| ?- wrong_ribua.
| 913.
| stop.


[ Error in arithmetic expression ]


[execution aborted]

| ?-
```

איור 12.3 – הפרוצדורה wrong_ribua

הבעיה העיקרית של הפרוצדורה wrong_ribua הינה האובדן של המספר המוקש, כאשר המטרה read(stop) נכשלת. הפרוצדורה ribua, שמוצגת באיור 12.4, פותרת את הבעיה הזאת כך: אחרי קריאת הביטוי המוקש היא מזמינה את הפרוצדורה proceed. בדרך זו שומרים על הביטוי המוקש.

```
ribua :- write('Please, type a positive integer'),
         nl , nl , tab(10),
         write('(to finish the game type stop)'),
         nl , nl, tab(5),
         read(N), proceed(N).
proceed(stop) :- !.
proceed(N) :- R  is  N * N , write('The square of  '),
              write(N), write(' equals '),
              write(R) , nl , nl , tab(2) , ribua.
yes
| ?- ribua.
Please, type a positive integer

          (to finish the game type stop)

     7.
The square of  7 equals 49

  Please, type a positive integer

          (to finish the game type stop)

     913.
The square of  913 equals 833569

  Please, type a positive integer

          (to finish the game type stop)

     stop.
```

איור 12.4 – השימוש ב-read

בכתיבת הפרוצדורה ribua השתמשנו בפרדיקטים nl ו-tab, כדי לערוך את תבנית התשובה.

גישה לתווים, Cut (איורים 12.5, 12.6, 12.7, 12.8)

נעבור לפרדיקטים get, put ו-name. הפרוצדורה go של הסעיף הקודם מתארת את פעולתם של הפרדיקטים get ו-put. פרוצדורה זו מוצגת באיור 12.5. הפרוצדורה "דוחקת" את הביטוי שהתקבל, ומחזירה אותו ללא רווחים.

```
go :- get(C),
      put(C),
      go_on(C).

go_on(46) :- !.
go_on(C) :- go.


| ?- go.
| to be  or   not     to      be
tobeornottobe|
| 1 2  3   4    5     6      7  8 9 0
1234567890|
```

איור 12.5 – פרדיקטים put, get

את השימוש בפרדיקטים put, name נדגים באמצעות הפרוצדורה starts, המתוארת באיור 12.6. הפרדיקט name מתרגם את המילה word לרשימת קודים ASCII, והפרדיקט put מציין את הראשון שבהם. נעיר כי הדפסת התו h נעשית מייד לאחר קריאה של המטרה put(104).

```
starts(Word, Character) :-
                    name(Word, [Character|Tail]),
                    put(Character).

| ?- starts(hello, ASCII_code).
h
ASCII_code = 104

yes
| ?- trace.
tracing: | ?- starts(hello, ASCII_code).
[0] call: starts(hello,ASCII_code81)?
 [1] call: name(hello,[Character173|Tail175])?
 [1] exit: name(hello,[104,101,108,108,111])?
 [1] call: put(104)?
h [1] exit: put(104)?
[0] exit: starts(hello,104)?
... in 1 reductions.

ASCII_code = 104
```

איור 12.6 – פרדיקטים namc, put

לפרדיקט name ישנם שני שימושים. האחד – לפרק לתווים את האטום הנתון (בדרך זו השתמשנו ב-name בפרוצדורה starts). השימוש האחר הינו למצוא את הביטוי על פי הרשימה הנתונה (את השימוש הזה של name ראינו באיור 12.2).

האיור 12.7 מתאר את השימוש בפרדיקט המערכת name שנעשה בפרוצדורה disjoint_characters. המטרה disjiont_characters מצליחה, אם שני האטומים הנתונים Atom1 ו-Atom2 מורכבים מקבוצות זרות של תווים.

```
disjoint_characters(Atoml,Atom2) :-
   name(Atoml,Listl),
   name(Atom2,List2),
    not (member(X,Listl) , member(X,List2)).


tracing: ¦ ?- disjoint_characters(atom,turbo).
[0] call: disjoint_characters(atom,turbo)?
 [1] call: name(atom,Listl171)?
 [1] exit: name(atom,[97,116,111,109])?
 [1] call: name(turbo,List2172)?
 [1] exit: name(turbo,[116,117,114,98,111])?
 [1] call: member(X175,[97,116,111,109])?
 [1] exit: member(97,[97,116,111,109])?
 [1] call: member(97,[116,117,114,98,111])?
 [1] fail: member(97,[116,117,114,98,111])?
 [1] redo: member(97,[97,116,111,109])?
 [1] exit: member(116,[97,116,111,109])?
 [1] call: member(116,[116,117,114,98,111])?
 [1] exit: member(116,[116,117,114,98,111])?
 [1] redo: name(turbo,[116,117,114,98,111])?
 [1] fail: name(turbo,_172)?
 [1] redo: name(atom,[97,116,111,109])?
 [1] fail: name(atom,_171)?
[0] fail: disjoint_characters(atom,turbo)?

no
```

איור 12.7 – הפרוצדורה disjoint_characters ללא Cut

הבה נעקוב אחרי חישוב המטרה disjoint_characters(atom, turbo). לאחר שמתגלה כי התו 116 שייך לשתי הרשימות, המטרה נכשלת. בשלב זה המערכת נסוגה, כדי לנסות למצוא פתרון אלטרנטיבי. היא בודקת האם יש פתרון אחר למטרה name(turbo, List2) וכשמוצאת כי אין – היא נסוגה צעד נוסף ובוחנת את name(atom, List1). המתכנת יודע שלמטרות name(turbo,List2) ו-name(atom,List1) יש פתרון אחד ויחיד. לכן הנסיגה מיותרת וכדי למנוע אותה, המתכנת מוסיף Cut אחרי name(Word2, List2). את הפרוצדורה שהתקבלה ואת ה-trace שלה ניתן לראות באיור 12.8.

```
disjoint_char(Word1,Word2) :-
   name(Word1,List1),
   name(Word2,List2),
   !,
    not (member(X,List1) , member(X,List2)).


tracing: ¦ ?- disjoint_char(atom,turbo).
[0] call: disjoint_char(atom,turbo)?
 [1] call: name(atom,List1171)?
 [1] exit: name(atom,[97,116,111,109])?
 [1] call: name(turbo,List2172)?
 [1] exit: name(turbo,[116,117,114,98,111])?
 [1] call: member(X175,[97,116,111,109])?
 [1] exit: member(97,[97,116,111,109])?
 [1] call: member(97,[116,117,114,98,111])?
 [1] fail: member(97,[116,117,114,98,111])?
 [1] redo: member(97,[97,116,111,109])?
 [1] exit: member(116,[97,116,111,109])?
 [1] call: member(116,[116,117,114,98,111])?
 [1] exit: member(116,[116,117,114,98,111])?
[0] fail: disjoint_char(atom,turbo)?

no
```

איור 12.8 – הפרוצדורה disjoint_characters עם Cut

גישה לביטוים מורכבים, פניה לפיסקאות (איורים 12.9, 12.10)

איור 12.9 מתאר את פעולת הפרדיקטים functor ו-clause. היחס functor(Term, Functor, Arity) מצליח, כאשר ביטוי מוצב תחת המשתנה Term, או כאשר נתונים הפנקטור והאריטי. דוגמה נוספת לפעולת הפרדיקט הזה נראה באחד השאלות בסעיף 12.4. המטרה clause(Head, Body) מצליחה, כאשר הראש של הפיסקה מוצב תחת המשתנה Head.

איור 12.9 –

```
yes
¦ ?- functor(disjoint_char(X,Y), F, A).

X = X28
Y = Y29
F = disjoint_char
A = 2

yes
¦ ?- functor(Term, disjoint_char, 2).

Term = disjoint_char(_42,_43)
```

```
yes
| ?- clause(disjoint_char(X,Y), Body).

X = Word159
Y = Word260
Body = name(Word159,List161) , name(Word260,List262) ,
! ,  not (member(X65,List161) , member(X65,List262))


yes
| ?- clause(go, Body).

Body = get(C51) , put(C51) , go_on(C51)

yes
| ?- clause(go_on(X), Body).

X = 46
Body = ! ;

X = C55
Body = go ;

no
| ?-
```

איור 12.9 – הפרדיקטים functor ו-clause

איור 12.10 מתאר את פעולת הפרדיקטים listing, assert, retract, abolish. לאחר הטענת המערכת בודקים אם היחס parent מוגדר, או לא. הבדיקה נעשית באמצעות השאילתה listing(parent). היחס איננו מוגדר. בהמשך מגדירים את היחס באמצעות assert, ומוחקים אותו באמצעות retract ו-abolish. התשובה לשאילתה אחרונה listing(parent) מגלה, כי אכן חזרנו למצב, שבו היינו כאשר הטענו את המערכת.

איור 12.10 –

```
A>wisdom
Wisdom(tm) Prolog interpreter version 2.10
Copyright (C) M. Safra & M.Goldberg(MAXBYTE), Israel
Copyright (C) MIT Press (U.S.A)

yes
| ?- listing(parent).


yes
| ?- assert(parent(tom,dick)).

yes
| ?- assert(parent(sara,dick)).

yes
| ?- listing(parent).

parent(tom,dick).
parent(sara,dick).
```

```
yes
| ?- asserta(parent(gloria,tom)).

yes
| ?- listing(parent).

parent(gloria,tom).
parent(tom,dick).
parent(sara,dick).


yes
| ?- assertz((parent(Parent,Child) :- same_address(Parent,
|                Child), same_family(Parent, Child))).

Parent = Parent29
Child = Child30

yes
| ?- listing(parent).

parent(gloria,tom).
parent(tom,dick).
parent(sara,dick).
parent(Parent,Child) :-
   same_address(Parent,Child),
   same_family(Parent,Child).


yes
| ?- retract(parent(tom, dick)).

yes
| ?- listing(parent).

parent(gloria,tom).
parent(sara,dick).
parent(Parent,Child) :-
   same_address(Parent,Child),
   same_family(Parent,Child).

yes
| ?- abolish(parent,2).

yes
| ?- listing(parent).


yes
| ?-
```

איור 12.10 – הפרדיקטים abolish, retract ,assert ,listing

קריאת קבצים (איור 12.11)

הטענת קובץ עם עובדות וחוקים נעשית באמצעות הפרדיקטים consult ו-reconsult, כפי שהודגם באיור 12.11.

```
Wisdom(tm) Prolog interpreter version 2.10
Copyright (C) M. Safra & M.Goldberg(MAXBYTE), Israel
Copyright (C) MIT Press (U.S.A)

yes
¦ ?- consult(subterm).
subterm   consulted

yes
¦ ?- [ribua, +go].
ribua   reconsulted
go   consulted
```

איור 12.11 – פרדיקטים consult, reconsult

בקרת זרימה (איור 12.12)

הדוגמה הבאה מתארת את בקרת הזרימה באמצעות הפרדיקטים repeat ו-fail. הפרוצדורה echo המוצגת באיור 12.12, קוראת ביטוי, כותבת אותו מייד, קוראת את הביטוי הבא, כותבת אותו, וחוזר חלילה. הזרימה מנוהלת באמצעות הפרדיקטים repeat ו-fail. המערכת איננה מאפשרת להגדיר את הפרדיקט repeat, ולכן "רימינו" אותה והגדרנו rrepeat, כפי שהגדרנו את הפרוצדורות mmember ו-nnot בפרק 11.

נעקוב באמצעות ה-trace אחרי חישוב המטרה echo. התכנית קולטת את המספר 2 וכותבת אותו. לאחר מכן היא מבצעת את המטרה nl. בתרשים של ה-trace הקריאה וההצלחה של המטרה nl מתבטאות על ידי קפיצת סימן השאלה לשורה החדשה. לאחר מכן בא תורה של המטרה fail, אשר כמובן נכשלת. כתוצאה מזה מתבצעת נסיגה. תוך כדי נסיגה חוזרת התכנית אל המטרה repeat, אשר מצליחה, וחוזר חלילה.

```
echo :-
   rrepeat,
   read(X),
   echo(X),
   !.
echo(end) :-
   !.
echo(X) :-
   write(X),
   nl,
   fail.
rrepeat.
rrepeat :-
   rrepeat.

yes
¦ ?- echo.
¦ tom.
tom
¦ liz.
liz
¦ loves(tom,liz).
loves(tom,liz)


yes
¦ ?- trace.
tracing: ¦ ?- echo.
[0] call: echo?
 [1] call: rrepeat?
 [1] exit: rrepeat?
 [2] call: read(X167)?
¦ 2.
 [2] exit: read(2)?
 [2] call: echo(2)?
  [3] call: write(2)?
2  [3] exit: write(2)?
  [3] call:
?

  [3] exit:
?
  [3] call: fail?
  [3] fail: fail?
  [3] redo:
?
  [3] fail:
?
```

(המשך)

```
  [3] redo: write(2)?
  [3] fail: write(2)?
 [2] fail: echo(2)?
 [2] redo: read(2)?
 [2] fail: read(_167)?
 [1] redo: rrepeat?
  [2] call: rrepeat?
  [2] exit: rrepeat?
 [1] exit: rrepeat?
 [3] call: read(_167)?
¦ liz.
 [3] exit: read(liz)?
 [3] call: echo(liz)?
  [4] call: write(liz)?
liz  [4] exit: write(liz)?
  [4] call:
?

  [4] exit:
?
  [4] call: fail?
  [4] fail: fail?
  [4] redo:
?
  [4] fail:
?
  [4] redo: write(liz)?
  [4] fail: write(liz)?
 [3] fail: echo(liz)?
 [3] redo: read(liz)?
 [3] fail: read(_167)?
 [1] redo: rrepeat?
  [2] redo: rrepeat?
   [3] call: rrepeat?
   [3] exit: rrepeat?
  [2] exit: rrepeat?
 [1] exit: rrepeat?
 [4] call: read(_167)?
¦ end.
 [4] exit: read(end)?
 [4] call: echo(end)?
 [4] exit: echo(end)?
[0] exit: echo?
... in 5 reductions.

¦ ?-
```

איור 12.12 – התכנית echo עם trace

12.3 פרדיקטים אקסטרה-לוגיים ומטה-לוגיים

בין פרדיקטי המערכת ניתן להבחין בפרדיקטים משני סוגים. להלן נביא את התאור והמאפיינים של הסוגים האלה.

12.3(א) פרדיקטים אקסטרה-לוגיים (extralogical)

סוג זה של פרדיקטים חורג ממסגרת המודל של תכנות לוגי. שפת הפרולוג, אשר באמצעותה מוצג הרעיון של תכנות לוגי, הינה שפה הצהרתית, הבנויה על בסיס לוגיקת היחסים. השפה מתארת את העצמים והקשרים ביניהם. לא נדרש מן המשתמש לחשוב במונחים של ביצוע ההוראות, כדי להבין מה עושה התכנית ואיך היא עושה זאת. את תהליך החישוב, כלומר, את ה-"איך", ניתן לראות כתהליך הסקה לוגית.

את שפת הפרולוג, אשר בנויה על-פי העקרונות של תכנות לוגי, מכנים בשם **פרולוג צרוף** (pure PROLOG). לתכנית של פרולוג צרוף, המבצעת חיפוש אחר המטרה ונסיגות, דרושים משאבים רבים. כדי לייעל את פעולת התכנית, מוסיפים פרדיקטים (כמו למשל, את הפרדיקט Cut), אשר אינם עומדים בקנה אחד עם הרעיון של תכנות לוגי (למשל, בתכנית עם Cut יש לעתים חשיבות לסדר של כתיבת פיסקאות, ראה איורים 11.3 ו-11.4 בפרק 11).

ישנן סיבות נוספות שלא לתכנת בפרולוג צרוף. למחשב ישנם אמצעים לבצע פעולות רבות כמו למשל, פעולות החשבון ופעולות קלט/פלט. שפת תכנות, אשר מטרתה להיות שימושית ויעילה, חייבת להיעזר בהם. ניתן למשל, להגדיר את פעולות החשבון באופן התואם את הרעיון של תכנות לוגי (ראה סעיף 4 של חטיבה רביעית), אבל בשל היעילות מעדיפים לוותר על כך ולהשתמש בשגרות (routines) של המחשב. פעולות קלט/פלט אינן כלולות במודל של תכנות לוגי, המיושם במחשב סדרתי. נציין, כי נעשים נסיונות להכלילם במודל של תכנות לוגי במסגרת הפרולוג המקבילי [Shapiro, 1983].

נביא מספר דוגמאות של התנהגות חורגת, שעל פיה נסווג את הפרדיקט כאקסטרה-לוגי. נתחיל מפרדיקטים המטפלים בקלט/פלט. אחד המאפיינים של הפרדיקטים האלה הינו היעדר פתרונות אלטרנטיביים. כלומר, הם מצליחים או נכשלים פעם אחת בלבד.

- **read, get0, get** – באמצעות הפרדיקטים האלה המערכת קוראת נתונים מתוך קובץ מסוים. קובץ זה מכונה בשם **זרם הקלט הנוכחי** (current input stream). לוח המקשים לצורכי הקריאה מסווג כקובץ, אשר כינויו user. הקבצים הינם קבצי ASCII, כלומר, קבצים של תווים, וניתן לקרוא אותם באופן סדרתי בלבד. לאחר שהמערכת קראה נתון מסויים, ההוראה read הבאה תגרום לקריאת הנתון שבא אחריו, וכן הלאה. לא ניתן לחזור ולקרוא את הנתון הקודם. חריגה זו הודגמה בפרוצדורה wrong_ribua.

- **see** – ניתן לקרוא את הנתונים מתוך הקובץ File באמצעות הפרדיקט see(File). לאחר ההטענה, המערכת מוכנה לקרוא מתוך הקובץ user, אשר מוגדר אוטומטית כזרם הקלט הנוכחי. כשרוצים לשנותו, נעזרים בפרדיקט see(File), אשר מעביר את זרם הקלט הנוכחי מקובץ user לקובץ File. המטרה see(File) מצליחה פעם אחת. בעת נסיגה, המערכת מדלגת על המטרה see(File) כלומר, זרם הקלט הנוכחי אינו משתנה.

- **seen** – סוגר את הקובץ ששימש כזרם הקלט הנוכחי, ומחזיר את המערכת לקובץ user.

- **put ,write** – באמצעות הפרדיקטים האלה ניתן לכתוב את הנתונים אל תוך הקובץ, המכונה בשם **זרם הפלט הנוכחי** (current output stream). המסך מסווג לצורך זה כקובץ בשם user.

- **tell** – באמצעות הפרדיקט tell(File) פותחים את זרם הפלט הנוכחי. המטרה tell(File) מצליחה פעם אחת בלבד, ואינה נכשלת בעת הנסיגה.

- **told** – הפרדיקט הזה סוגר את זרם הפלט הנוכחי.

נביא דוגמאות אחרות של התנהגות חורגת, אשר מקורן בהצבות תחת המשתנה.

- **not** – התנהגותו של הפרדיקט הזה חורגת ממסגרת תכנות לוגי, כאשר בעת הפעלתו נמצאים בתחומו משתנים שאינם קשורים. דוגמאות של סטיות מסוג זה מובאות בסעיף 11.3. לעומת זה, כאשר ההצבות תחת המשתנים אשר בתחומו נעשו לפני הפעלתו, הוא מתנהג כפרדיקט לוגי לכל דבר, [Jaffar & others, 1983]. בסעיף 11.3 הדגמנו את הבדלי התנהגותו של not באמצעות הפרוצדורות unhappy ו-unhappy1.

- **assert** – היכרנו את אופן השימוש בפרדיקט זה בסעיף הקודם. בלעדיו לא ניתן לבנות מערכת מומחה (ראה חטיבה רביעית).
נציין כמה מתכונות הפרדיקט, לאחר עיון בשתי השאילתות הבאות:

```
?- assert(sunny), sunny.
?- sunny, assert(sunny).
```

התשובה לשאילתה הראשונה הינה חיובית. התשובה לשאילתה השניה תלויה בכך, שהעובדה sunny נמצאת, או שאינה נמצאת, בבסיס הנתונים. האופרטור "," אמור לשמש כקשר לוגי AND. בהשפעת הפרדיקט assert הוא אינו ממלא תפקיד זה, שהרי על פי תכונת השקילות שלו (ראה פרק 2), הפסוק a AND b שקול לפסוק b AND a, דבר שאינו נכון במקרה של שתי השאילתות שלעיל.

נעיין בשתי שאילתות נוספות:

```
?- happy(X), clever(X).
?- assert(happy(X)), clever(X).
```

על פי העקרון של תכנות לוגי, המשתנה X בשאילתה המורכבת הינו משתנה משותף. העקרון הזה הופר בשאילתה השניה. משמעות הפיסקה ((assert(happy(X היא, שהעובדה happy(X) מצטרפת לבסיס הנתונים. אשר על כן המשתנה X בפיסקה ((assert(happy(X הינו משתנה המכומת כללי. לעומת זאת, אותו משתנה בפיסקה clever(X) מכומת קיומי, כנהוג במשתני השאילתה.
נעיין בשאילתה נוספת:

```
?- assert(happy(Y)), clever(X).
```

המשמעות של השאילתה הזו הינה זהה למשמעות של השאילתה הקודמת. למדנו מכך, כי כללי התכנית הלוגית בנוגע למשתנים משותפים בשאילתות מורכבות, כפי שהיכרנו אותם בפרק 5, מופרים על ידי השימוש ב-assert.

נציין תכונה נוספת של assert: הפיסקה שנוספה לבסיס הנתונים באמצעות assert, נשארת בו כשמתבצעת נסיגה. נציין גם, כי נעשים מאמצים לפתח שיטות, אשר יאפשרו להוסיף מידע לתכנית מבלי לפגוע בתכונות הלוגיות שלה [Warren, 1984].

ישנם פרדיקטים אקסטרה-לוגיים נוספים. את השאלה אם הפרדיקט הנתון הינו אקסטרה-לוגי אם לאו, פותרים לגבי כל פרדיקט. בחרנו לסווג את הפרדיקטים בהתאם להתנהגותם. חשוב ללמוד את התנהגותם כדי לדעת לאתר סטיות ובכך למנוע את סילופי המשמעות בתכנית לוגית.

12.3(ב) פרדיקטים מטה-לוגיים (metalogical)

תחילה נדון במונח "מטה" (meta), שתרגומו לעברית הינו "על". אם-כן, תרגומו של המונח metaprogram הינו "תכנית-על". תכנית-על היא תכנית אשר הקלט שלה הוא תכנית אחרת. למשל, המהדר (compiler) הינו דוגמה של תכנית-על.

פרדיקט מטה-לוגי הינו פרדיקט המערכת, אשר יעודו לטפל בפרדיקטים אחרים. פרוש הדבר הוא, שהארגומנטים של פרדיקט מטה-לוגי הינם פיסקאות של תכנית לוגית. נביא כמה דוגמאות של פרדיקטים מסוג זה.

- **call** – התכנית של פרולוג הינה אוסף הנתונים. הפרדיקט call(X) "הופך" את הנתון X למטרה X ובכך מתאפשרת ריצת התכנית. הכלל שלפיו מחשבים את המטרה call(X) הינו כדלהלן: המטרה call(X) מצליחה, אם המטרה X מצליחה, והמטרה call(X) נכשלת אם המטרה X נכשלת. את השימוש בפרדיקט המערכת call ראינו בעת ביצוע המעקב (trace). בתור דוגמה נוספת נביא את ההגדרה הבאה:

```
not Goal :- call(Goal), !, fail.
not Goal.
```

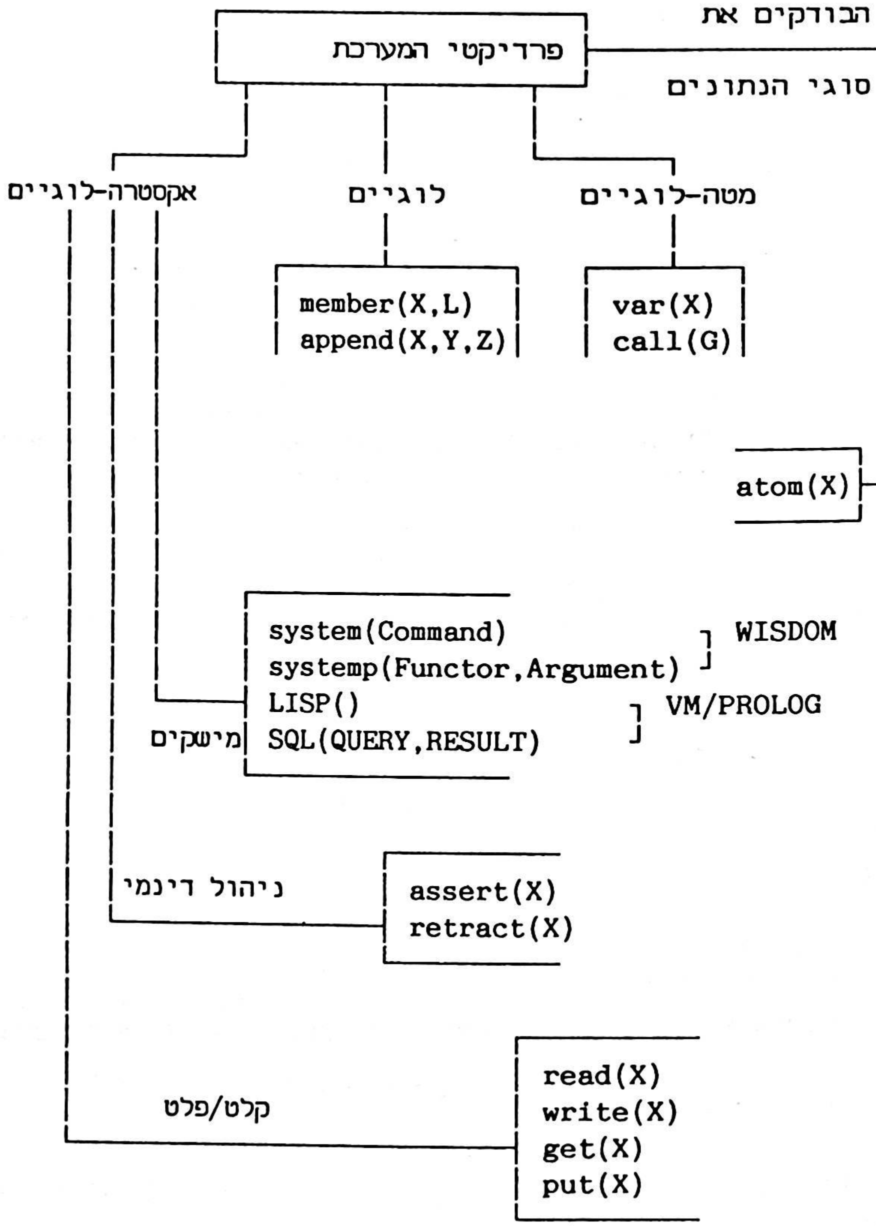
תכנית 12.8 – שימוש בפרדיקט call

- **var** – הפרדיקט var(Term) מבצע בדיקה של סוג הנתון Term. המטרה var(Term) מצליחה, כאשר המשתנה Term אינו קשור. אם הנתון הינו ביטוי מורכב, המטרה var(Term) נכשלת.

השימוש בפרדיקטים מטה-לוגיים מהווה אמצעי יעיל בבניית קונכיות (shells). לנושא זה נחזור בחטיבה הרביעית.

סוגים שונים של פרדיקטי המערכת תוארו באיור 12.13. נציין, כי אין דעה אחידה לגבי החלוקה של הפרידקטים לאקסטרא-ומטה-לוגיים. ישנם פרדיקטים כמו למשל, פרדיקטי קלט/פלט, שהינם פרדיקטים אקסטרא-לוגיים ואין ספק לגבי השתייכותם. לעומתם, ישנם פרדיקטים (כמו למשל, not) אשר סיווגם איננו חד-משמעי (את הפרדיקט not מסווגים לפעמים כמטה-לוגי, כיוון שהוא מטפל בפסוקים, שהינם חלק של תכנית לוגית).

ספקות דומים קיימים גם לגבי הפרדיקטים שבודקים את סוג הנתון. ישנם שמסווגים את integer(X) כפרדיקט מטה-לוגי, בתנאי שבמקרים בהם X איננו קשור, מתקבלת תשובה שלילית, ולא הודעת השגיאה. בכל מקרה, מבחינה מעשית יש ללמוד היטב את פעולתם של פרדיקטי המערכת, כדי לדעת את מגבלותיהם ואת גבולות השימוש בהם.



איור 12.13 – פרדיקטי המערכת

12.4 פרדיקטי המערכת – תרגול

שאלה 1

א. הצג לפרולוג אשר ברשותך את השאילתות של איור 12.1, בדוק את התשובות והשווה את התשובות עם אלו שבאיור.

ב. הצג לפרולוג את השאילתות שלהלן ועיין בתשובות.

```
?- atom(parent(abraham, isaac)).

?- integer(parent(abraham, isaac)).

?- var(parent(abraham, isaac)).
```

הערה: באמצעות שאלה זו תוכל ללמוד לאיזו התנהגות יש לצפות מתכנית הפרולוג, אשר מכילה את הפרדיקטים לבדיקת סוג הנתונים.

שאלה 2

עיין בתכנית ribua אשר באיור 12.4. שנה את הפרוצדורה proceed כך, שלאחר הקשת הקלט תתבצע בדיקת סוג הנתון, ותשלח הודעה מתאימה, אם הקלט איננו מספר ואיננו האטום stop.

שאלה 3

הגדר את הפרוצדורה writelist(List) כך, שניתן יהיה לקבל את התשובה הבאה לשאילתה, אשר להלן:

```
?- writelist([a, b, c]).
a
 b
  c
```

שאלה 4

הגדר את הפרוצדורה stars(L) כך, שניתן יהיה לקבל את התשובה הבאה לשאילתה, אשר להלן:

```
?- stars([2, 5, 3]).
**
*****
***
```

שאלה 5

הגדר את הפרוצדורה prefix(Word1, Word2), אשר באמצעותה ניתן יהיה לבדוק האם המילה הנתונה Word1 הינה קידומת של המילה הנתונה Word2.
רמז: היעזר בפרדיקטים name ו-append.

```
| ?- disjoint_characters(you,sweet).

yes
| ?- trace.
tracing: | ?- disjoint_characters(you,sweet).
[0] call: disjoint_characters(you,sweet)?
 [1] call: name(you,List1171)?
 [1] exit: name(you,[121,111,117])?
 [1] call: name(sweet,List2172)?
 [1] exit: name(sweet,[115,119,101,101,116])?
 [1] call: member(X175,[121,111,117])?
 [1] exit: member(121,[121,111,117])?
 [1] call: member(121,[115,119,101,101,116])?
 [1] fail: member(121,[115,119,101,101,116])?
 [1] redo: member(121,[121,111,117])?
 [1] exit: member(111,[121,111,117])?
 [1] call: member(111,[115,119,101,101,116])?
 [1] fail: member(111,[115,119,101,101,116])?
 [1] redo: member(111,[121,111,117])?
 [1] exit: member(117,[121,111,117])?
 [1] call: member(117,[115,119,101,101,116])?
 [1] fail: member(117,[115,119,101,101,116])?
 [1] redo: member(117,[121,111,117])?
 [1] fail: member(_175,[121,111,117])?
[0] exit: disjoint_characters(you,sweet)?
```

איור 12.14 — ה-trace של disjoint_characters

שאלה 6

עיין באיור 12.14 וענה על השאלות הבאות.

א. מהי התשובה לשאילתה:

```
?- disjoint_char(you, sweet).
```

ב. האם ה-trace של חישוב השאילתה שלעיל יהיה שונה מזה שבאיור 12.14? נמק את תשובתך.

הערה: את הפרוצדורות disjoint_characters ו-disjoint_char היגדרנו באיורים 12.7 ו-12.8.

שאלה 7

התבונן בתכנית הבאה:

```
play(Word, Answer) :-
                atom(Word),
                var(Answer),
                name(Word, List),
                name(Answer, [121,111,117,32|List]).
```

מצא את התשובות לשאילתות הבאות:

```
?- play(dummy, Answer).
?- X=dummy, play(X, Answer).
?- play(X, Answer), X=dummy.
```

שאלה 8

לפניך פרוצדורות think ו-go_ahead. מבלי להריץ את התכנית, הסבר מה תהיינה התשובות לשאילתות שלהלן.

```
think(Goal) :-
            clause(Goal, Body),
            go_ahead(Body).
go_ahead(true) :-
            tab(12),
            write('TRUE!'),
            !,
            nl.
go_ahead(Body) :-
            tab(8),
            write('IF'),
            nl,
            tab(10),
            write(Body).
?- think(member(a, [a,b,c])).
?- think(member(b, [a,b,c])).
?- think(play(dummy, Answer)).
?- think(play(dummy, why)).
```

שאלה 9

עיין בפרוצדורה learn שבאיור 12.15, וענה על השאלות האלו:

א. האם ישתנה ה-trace של הפרוצדורה learn בהשוואה ל-trace של echo, שנמצא באיור 12.12? אם אכן ישתנה, מה יהיה השינוי?

ב. נניח שמריצים את הפרוצדורה echo, כאשר זרם הפלט הנוכחי הינו קובץ בשם echo1. מה תהיה התוצאה?

ג. נניח שמריצים את הפרוצדורה learn, כאשר זרם הפלט הנוכחי הינו קובץ בשם learn1. מה תהיה התוצאה?

```
learn :-
   rrepeat,
   read(X),
   assert(X),
   echo(X),
   !.
echo(end) :-
   !.
echo(X) :-
   write(X),
   nl,
   fail.
rrepeat.
rrepeat :-
   rrepeat.
```

איור 12.15 – הפרוצדורה learn

שאלה 10

עיין בפרוצדורה subterm וב-trace שלה אשר באיור 12.16

בתשובה לשאילתה:·

```
?- subterm(likes(tom, sisters(ann, liz))).
```

מקבלים אוסף של תת-ביטויים של הביטוי. לעומת זאת, ה-trace מסתיים בפרדיקט fail. התשובה לשאילתה מסתתרת בתוך ה-trace, ואינה מופיעה לקראת סופו. אתר את שלבי קבלת התשובה בתוך ה-trace והסבר את דרך קבלתה.

```
subterm(Term) :-
   subterm(Subterm,Term),
   tab(12),
   write(Subterm),
   nl,
   fail.
subterm(Subterm,Term) :-
    not atomic(Term),
   functor(Term,F,N),
   subterm(N,Subterm,Term).
subterm(T,T).
subterm(N,Subterm,Term) :-
   N > 1,
   N1 is N - 1,
   subterm(N1,Subterm,Term).
subterm(N,Subterm,Term) :-
   arg(N,Term,Argument),
   subterm(Subterm,Argument).

yes
| ?- subterm(likes(tom,sisters(ann,liz))).
            tom
            ann
            liz
            sisters(ann,liz)
            likes(tom,sisters(ann,liz))

| ?- trace.
tracing: | ?- subterm(likes(tom,sisters(ann,liz))).
[0] call: subterm(likes(tom,sisters(ann,liz)))?
 [1] call: subterm(Subterm173,likes(tom,sisters(ann,liz)))?
  [2] call: atomic(likes(tom,sisters(ann,liz)))?
  [2] fail: atomic(likes(tom,sisters(ann,liz)))?
  [2] call: functor(likes(tom,sisters(ann,liz)),F273,N274)?
  [2] exit: functor(likes(tom,sisters(ann,liz)),likes,2)?
  [2] call: subterm(2,Subterm270,likes(tom,sisters(ann,liz)))?
   [3] call: 2 > 1?
   [3] exit: 2 > 1?
   [3] call: N1491 is 2 - 1?
   [3] exit: 1 is 2 - 1?
   [3] call: subterm(1,Subterm488,likes(tom,sisters(ann,liz)))?
    [4] call: 1 > 1?
```

איור 12.16 – ריצה של פרוצדורה subterm

```
    [4] fail: 1 > 1?
    [4] call: arg(1,likes(tom,sisters(ann,liz)),Argument815)?
    [4] exit: arg(1,likes(tom,sisters(ann,liz)),tom)?
    [4].call: subterm(Subterm813,tom)?
     [5] call: atomic(tom)?
     [5] exit: atomic(tom)?
    [4] exit: subterm(tom,tom)?
   [3] exit: subterm(1,tom,likes(tom,sisters(ann,liz)))?
  [2] exit: subterm(2,tom,likes(tom,sisters(ann,liz)))?
 [1] exit: subterm(tom,likes(tom,sisters(ann,liz)))?
 [5] call:             ?
           [5] exit:             ?
 [5] call: write(tom)?
tom [5] exit: write(tom)?
 [5] call:
?

 [5] exit:
?
    ≈≈≈≈≈
      [5] call: arg(2,sisters(ann,liz),Argument915)?
      [5] exit: arg(2,sisters(ann,liz),liz)?
      [5] call: subterm(Subterm913,liz)?
       [6] call: atomic(liz)?
       [6] exit: atomic(liz)?
      [5] exit: subterm(liz,liz)?
     [4] exit: subterm(2,liz,sisters(ann,liz))?
    [3] exit: subterm(liz,sisters(ann,liz))?
   [2] exit: subterm(2,liz,likes(tom,sisters(ann,liz)))?
 [1] exit: subterm(liz,likes(tom,sisters(ann,liz)))?
 [6] call:             ?
           [6] exit:             ?
 [6] call: write(liz)?
liz [6] exit: write(liz)?
 [6] call:
?

    ≈≈≈≈≈

    [3] exit: subterm(sisters(ann,liz),sisters(ann,liz))?
   [2] exit: subterm(2,sisters(ann,liz),likes(tom,sisters(ann,liz)))?
 [1] exit: subterm(sisters(ann,liz),likes(tom,sisters(ann,liz)))?
 [4] call:             ?
           [4] exit:             ?
 [4] call: write(sisters(ann,liz))?
sisters(ann,liz) [4] exit: write(sisters(ann,liz))?
 [4] call:
?
```

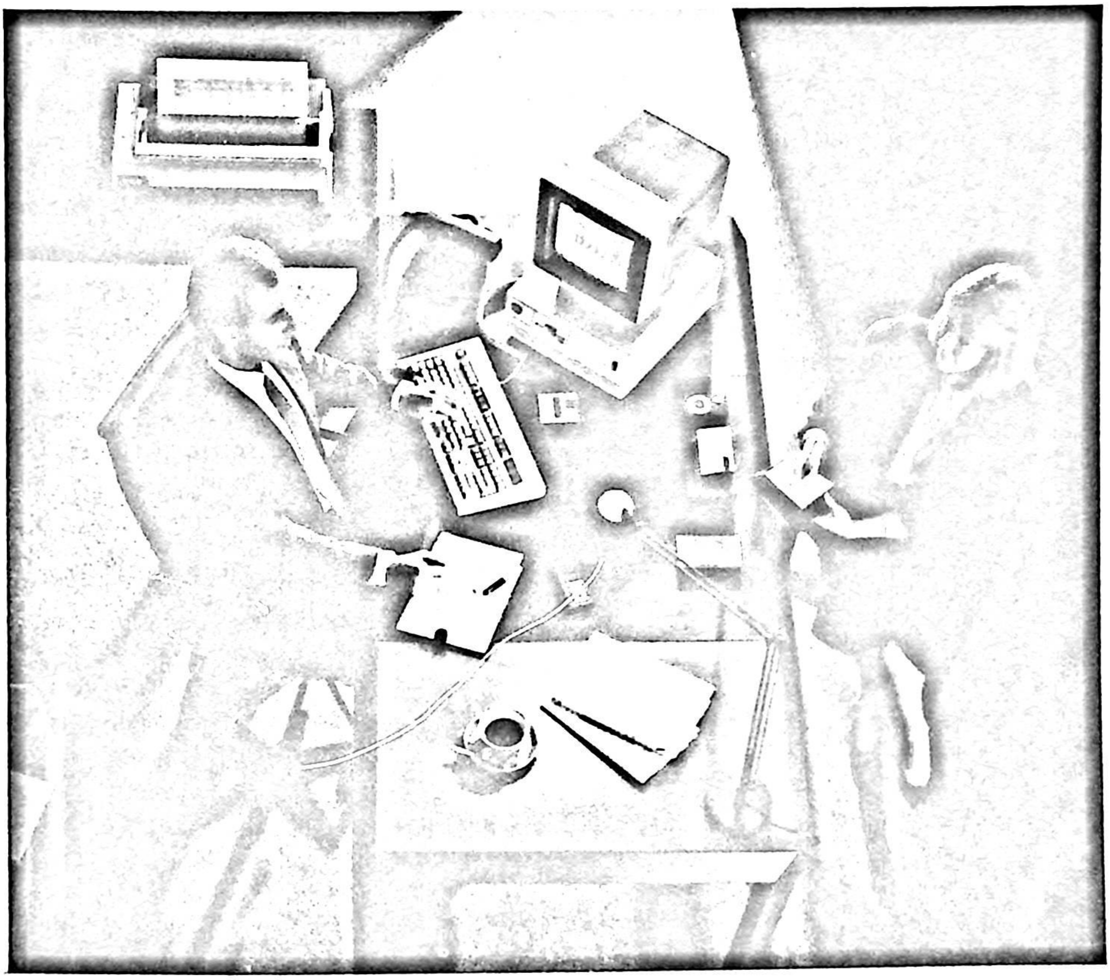
איור 12.16 – ריצה של פרוצדורה subterm (המשך 1)

```
[1] exit: subterm(likes(tom,sisters(ann,liz)),likes(tom,sisters(ann,liz)))?
[2] call:               ?
          [2] exit:             ?

 [2] redo: functor(likes(tom,sisters(ann,liz)),likes,2)?
 [2] fail: functor(likes(tom,sisters(ann,liz)),_273,_274)?
[1] exit: subterm(likes(tom,sisters(ann,liz)),likes(tom,sisters(ann,liz)))?
[2] call:               ?
          [2] exit:             ?
[2] call: write(likes(tom,sisters(ann,liz)))?
likes(tom,sisters(ann,liz)) [2] exit: write(likes(tom,sisters(ann,liz)))?
[2] call:
?

[2] exit:
?
[2] call: fail?
[2] fail: fail?
[2] redo:
?
[2] fail:
?
[2] redo: write(likes(tom,sisters(ann,liz)))?
[2] fail: write(likes(tom,sisters(ann,liz)))?
[2] redo:               ?
[2] fail:               ?
[1] redo: subterm(likes(tom,sisters(ann,liz)),likes(tom,sisters(ann,liz)))?
[1] fail: subterm(_173,likes(tom,sisters(ann,liz)))?
[0] fail: subterm(likes(tom,sisters(ann,liz)))?
```

איור 12.16 – ריצה של פרוצדורה subterm (המשך 2)



חטיבה רביעית

תכניות יישום

בחטיבה זו הצבנו שתי מטרות. האחת – להכיר את האפשרויות של פרולוג בנושא היישומים. לשם כך הצגנו סדרה של תכניות יישום בתחומים שונים. המטרה האחרת – לערוך חזרה על חומר הלימוד. לשם כך, ערכנו את התכניות לפי סדר הפרקים בספר, אשר ההיכרות אתם חיונית לכתיבת התכניות. מספרי הפרקים צויינו במבוא לכל תכנית יישום, בסעיף **"רקע"**. כל תכנית מלווה בתרגיל, אשר גם הוא מתאים לחומר הרקע. לארגון זה יתרון נוסף: ניתן לדון בתכנית היישום מיד לאחר הלימוד של הפרקים המתאימים.

לסיום החטיבה הצגנו מספר שאלות, אשר בהן הלומד נדרש להדגים ידע משולב של חומר הלימוד. דגש מיוחד ניתן לשאלות בעלות תוכן לימודי בתחומים שונים, כגון תנ"ך, הסטוריה, מתמטיקה וכדומה.

רשימת תכניות היישום של חטיבה זו:

1. סיפור מן החיים בשפת פרולוג.
   רקע: פרקים 5, 7 (בלי רקורסיה).
2. חד-גדיא.
   רקע: פרקים 5, 7.
3. קשרי המשפחה המנומקים.
   רקע: פרקים 5, 7 (בלי רקורסיה), 8 (בלי פעולות החשבון).
4. תאור של מבנים אינסופיים באמצעות רקורסיה.
   רקע: פרקים 5, 7, 8 (בלי פעולות החשבון).
5. חישוב עצרת של מספר.
   רקע: פרקים 5, 7, 8.

6. פתרון בעיה באמצעות תכנות לוגי – מפת היעדים.
רקע: פרקים 5, 7, 8 (בלי פעולות החשבון).
7. מודל לפתרון בעיה באמצעות תכנות לוגי.
רקע: פרקים 5, 7, 8.
8. חיפוש הדרך במבוך.
רקע: פרקים 5, 7, 8, 9.
9. מודל לפתרון בעיה באמצעות תכנות לוגי (המשך).
רקע: פרקים 5, 7, 8, 9, 10 (פרק 10 רצוי, אולם אינו הכרחי).
10. בעית שמונה המלכות.
רקע: פרקים 5, 7, 8, 9.
11. מגדלי האנוי.
רקע: פרקים 5, 7, 8, 9, 12 (מתוך פרק 12 נדרשת ההיכרות עם היחס write).
12. תוכנה לימודית באמצעות פרולוג – התפתחות של בעלי-חיים.
רקע: פרקים 5, 7, 11, 12 (היכרות עם היחסים write ו-nl).
13. מערכת מומחה – גן החיות.
רקע: פרקים 5, 7, 8, 11, 12.
14. קונכיה (Shell) של מערכת מומחה.
רקע: פרקים 5, 7, 8, 9, 10, 11, 12.
15. תכניות יישום – תרגול.

1. סיפור מן החיים בשפת פרולוג

חומר רקע: פרקים 5, 7 (בלי רקורסיה).

תכנות לוגי הינו תכנות הצהרתי (דקלרטיבי). תכנית לוגית היא אוסף של עובדות וחוקים, המתארים אירוע כלשהו. העובדות מתארות את היחסים בין העצמים ואת תכונותיהם, והחוקים מגדירים יחסים חדשים על בסיס היחסים הקיימים. כל חוק מבטא קשר לוגי בין היחסים.

את הסיפור על אדון נחמד בשם צ׳רלס, כתבנו במקביל בשפה האנגלית ובשפת הפרולוג. להלן הסיפור.

A TRUE LOVE STORY

Once upon a time there was a gentleman... And there were women in his life... And those were his rules:

He liked beautiful ladies.
liked(X, Y):-lady(Y), beautiful(Y).

He was fond of tall ladies.
was_fond_of(X, Y):-lady(Y), tall(Y).

He prefered blond hair ladies.
prefered(X, Y):-lady(Y), blond_hair(Y).

He appreciated intelligent ladies.
appreciated(X, Y):-lady(Y), intelligent(Y).

He understood beautiful gossips.
understood(X, Y):-gossip(Y), beautiful(Y).

He hated intelligent gossips.
hated(X, Y):-gossip(Y), intelligent(Y).

Among those he prefered
he travelled with those who were intelligent.
travelled_with(X, Y):-prefered(X, Y), intelligent(Y).

Among those he appreciated
he was a friend of those who were beautiful.
friend_of(X, Y):-appreciated(X, Y), beautiful(Y).

Among those he prefered
he loved those who were beautiful.
loved(X, Y):- prefered(X, Y), beautiful(Y).

He married whom he loved and
who was intelligent.
married(X, Y):-loved(X, Y), intelligent(Y).

But he also was a close friend of whom he understood,
was fond of and prefered.
close_friend(X, Y):-understood(X, Y), was_fond_of(X, Y), prefered(X, Y).

And those were his ladies:

```
lady(sara).
lady(ester).
lady(ruth).
lady(alice).
lady(victoria).
lady(patricia).
lady(liz).
lady(nataly).
gossip(victoria).
gossip(mary).
gossip(liz).
gossip(patricia).
tall(ester).
tall(liz).
tall(mary).
tall(patricia).
beautiful(sara).
beautiful(ruth).
beautiful(liz).
beautiful(nataly).
blond_hair(sara).
blond_hair(victoria).
blond_hair(ester).
blond_hair(liz).
intelligent(sara).
intelligent(alice).
intelligent(nataly).
intelligent(mary).
```

תכנית 1 – סיפור מן החיים בשפת פרולוג

תרגיל 1

ידוע כי: Charles was the name of the gentleman

א. ענה על השאלות הבאות:

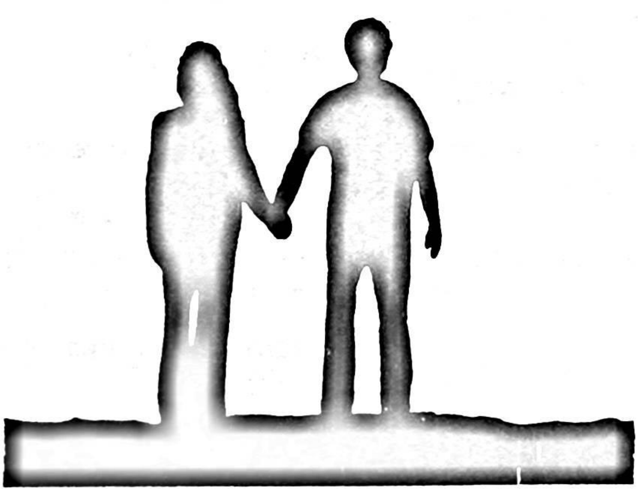
Whom did Charles like? prefer? appreciate?
Whom did he love? Who were his friends?
Whom did he travel with? Whom did he hate?
Whom was he fond of? Whom did he understand?
Whom did he married? And who were his close friends?

ב. תרגם את השאלות שבסעיף א' לשאילתות של פרולוג.

ג. הרץ את התכנית, והשווה את התשובות לשאילתות עם התשובות של סעיף א', אשר השבת על פי סיפור המעשה.

ד. המצא המשך ל"סיפור מן החיים". למשל: תאר את חיי משפחת צ'רלס עם בחירת ליבו, תאר את ילדיהם, או לחילופין, תאר את חבריו של צ'רלס, את עיסוקיהם וכן הלאה.

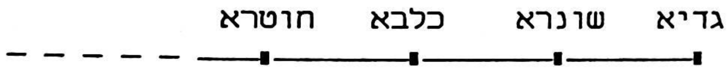
ה. כתוב סיפור בפרולוג בנושא האהוב עליך.



2. חד-גדיא

חומר רקע: פרקים 5, 7.

התכנית שלהלן הינה בעלת תוכן לימודי בנושא השיר חד-גדיא, שבהגדת חג הפסח. השיר תורגם לעובדות של פרולוג. את הקשרים בין העצמים המוזכרים בשיר, תיארנו באמצעות חוקים רקורסיביים. כדי לבנות את החוקים האלו, היה עלינו להפוך את סדר הזוגות, הקשורים באמצעות היחס "ואתא". מדוע? על שאלה זו תתבקש לענות בתרגיל שבהמשך.



איור 1 – שרשרת של חד-גדיא

להלן התכנית. באמצעותה ניתן לראות את הקשרים בין משתתפי סיפור החד-גדיא. קשרים נוספים בין המשתתפים ומעשיהם תתבקש לפתח באחת השאלות של סעיף 15.

```
ואתא(שונרא, אכל, גדיא).
ואתא(כלבא, נשך, שונרא).
ואתא(חוטרא, היכה, כלבא).
ואתא(נורא, שרף, חוטרא).
ואתא(מיא, כבה, נורא).
ואתא(תורא, שתא, מיא).
ואתא(שוחט, שחט, תורא).
ואתא(מלאך_המות, שחט, שוחט).
ואתא(הקדוש_ברוך_הוא, שחט, מלאך_המות).

זוג(_עצם1, _עצם2) :-
    ואתא(_עצם2, _מעשה, _עצם1).

שרשרת(_עצם1, _עצם2) :-
    זוג(_עצם1, _עצם2).
שרשרת(_עצם1, _עצם2) :-
    זוג(_עצם1, _עצם),
    שרשרת(_עצם, _עצם2).

קשר(_עצם1, _עצם2) :-
    שרשרת(_עצם1, _עצם2);
    שרשרת(_עצם2, _עצם1).

עושה(_עצם, _מעשה) :-
    ואתא(_עצם, _מעשה, _).
עושה(_עצם, _מעשה) :-
    ואתא(_עצם, _פעולה, _עצם1),
    עושה(_עצם1, _מעשה).
```

תכנית 2 – חד-גדיא

נציין, כי היחס **שרשרת** מתאר את הקשר בין העצמים על פי סדר הופעתם בשיר. לעומת זה, היחס **קשר** הינו יחס סימטרי: אם הוא מתקיים בין גדיא לבין שונרא, אז הוא מתקיים בין שונרא לבין גדיא, וכן הלאה.

את התכנית "חד-גדיא" ניתן לפתח בכיוונים שונים. נחזור אליה בשאלה 2 בסעיף 15.

תרגיל 2

א. לשם מה הפכנו את סדר הופעת העצמים באמצעות היחס **זוג**? האם ניתן לבנות את התכנית מבלי להפוך את היחס?
נמק את תשובתך.

ב. מצא את התשובות לשאילתות האלה:

?- שרשרת(גדיא, שוחט).
?- שרשרת(שוחט, גדיא).
?- קשר(גדיא, שוחט).
?- קשר(שוחט, גדיא).

ג. אילו ממשתתפי החד-גדיא קשורים אל חוטרא?

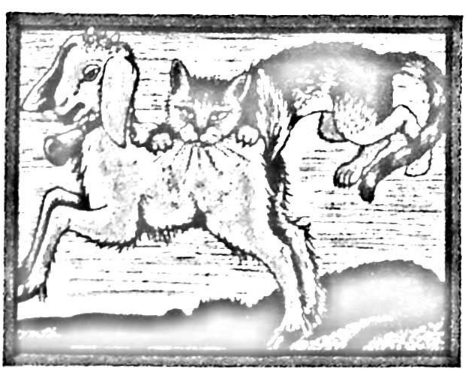
ד. התבונן בשאילתות הבאות:

?- שרשרת(נורא, X).
?- קשר(נורא, X).
?- עושה(נורא, X).
?- שרשרת(X, נורא).
?- קשר(X, נורא).
?- עושה(X, היכה).

מצא את כל התשובות של פרולוג לכל אחת מהן.

ה. הוכח באמצעות השאילתות והתשובות, כי הקדוש-ברוך-הוא קשור לכל אחד ממשתתפי חד-גדיא.

ו. הוכח באמצעות שאילתות ותשובות כי הקדוש-ברוך-הוא עושה את כל אחד ממעשיו של חד-גדיא.



3. קשרי המשפחה המנומקים

חומר רקע: פרקים 5, 7 (בלי רקורסיה), 8 (בלי פעולות החשבון).

את קשרי המשפחה פיתחנו בפרק 7 בתכנית Family Rules. עתה, ברצוננו לנמק את כל אחד מחוקי הקשר המשפחתי. נבטא זאת באמצעות היחס because.
שפת הפרולוג תומכת בנתון מבני, על-כן ניעזר בו לבניית קשר משפחתי מנומק. להלן התכנית:

```
parent(abraham, isaac, parent(abraham, isaac)).
parent(isaac, jacob, parent(isaac, jacob)).
parent(sarah, isaac, parent(sarah, isaac)).
parent(rivkah, jacob, parent(rivkah, jacob)).

male(abraham, male(abraham)).
male(isaac, male(isaac)).
male(jacob, male(jacob)).

female(sarah, female(sarah)).
female(rivkah, female(rivkah)).

father(X, Y, because(A, B)) :-
            parent(X, Y, A), male(X, B).
mother(X, Y, because(A, B)) :-
            parent(X, Y, A), female(X, B).
grandfather(X, Y, because(B, A, C)) :-
            father(X, Z, because(A, B)), parent(Z, Y, C).
grandmother(X, Y, because(B, A, C)) :-
            mother(X, Z, because(A, B)), parent(Z, Y, C).
couple(X, Y, because(A, B, C, D)) :-
            father(X, Z, because(A, B)), mother(X, Z, because
                                                        (C, D)).
child(X, Y, because(A)) :-
            parent(Y, X, A).
son(X, Y, because(A, B)) :-
            child(X, Y, because(A)), male(X, B).
daugther(X, Y, because(A, B)) :-
            child(X, Y, because(A)), female(X, B).
```

תכנית 3 – קשרי המשפחה המנומקים

לתכנית זו נחזור בשאלה 3 בסעיף 15.

תרגיל 3

א. מצא את כל התשובות לשאילתות הבאות:

```
?- father(X, Y, Z).
?- mother(X, Y, because(A, B)).
?- couple(X, Y, because(A, B, C, D)), son(X, Z, because(E, B)).
```

ב. תאר באמצעות עץ את הביטויים המורכבים האלה:

```
1. grandfather(X, Y, Z).
2. grandfather(X, Y, because(B, A, C)).
3. grandfather(abraham, jacob, because(male(abraham),
               parent(abraham, isaac), parent(isaac, jacob))).
```

ג. כתוב תכנית כלשהי עם הסברים (למשל, הוסף הסברים לתכנית "סיפור מן החיים" שבסעיף 1).

4. תאור של מבנים אינסופיים באמצעות רקורסיה

חומר רקע: פרקים 5, 7, 8 (בלי פעולות החשבון).

באמצעות הגדרה רקורסיבית ניתן לטפל במבנים אינסופיים, כמו מספרים טבעיים, או במבנים בעלי פוטנציאל אינסופי, כמו דורות של אבותינו.
נתבונן בתכנית הלוגית הבאה:

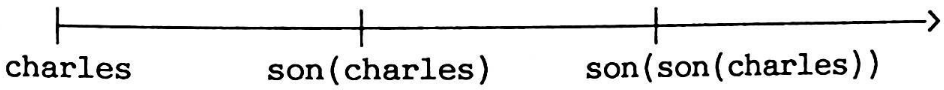
```
clever(charles).
clever(son(X)) :- clever(X).
```

תכנית 4(א) – צאצאיו של צ׳רלס

נציג שאילתה ונעיין בתשובות:

```
?- clever(X).
X = charles;
X = son(charles);
X = son(son(charles));
X = son(son(son(charles)));
```

וכן הלאה.
את התשובה הראשונה קיבלנו על פי החוק הראשון. את התשובות הנוספות קיבלנו באמצעות הרקורסיה. אם כן, התכנית הזאת מטפלת בסדרה אינסופית של ביטויים בסיסיים, אשר מוצגים באיור 2.



איור 2 – סדרת צאצאיו של צ׳רלס

ביטויים אלו מייצגים את מיודענו צ׳רלס של תכנית 1 ואת צאצאיו:

בנו של צ׳רלס – son(charles)
נכדו של צ׳רלס – son(son(charles))
נינו של צ׳רלס – son(son(son(charles)))
וכן הלאה.

לסדרה של מספרים טבעיים יש מבנה דומה לזה של סדרת צאצאיו של צ׳רלס. את המספר 0 נציב במקום "צ׳רלס". את המספר 1 ניתן, אם כן, לזהות עם בנו, את המספר 2 – עם נכדו, וכן הלאה. לצורך זיהוי זה, נסמן את המספר 1 באמצעות הסימון s(0). בהמשך נקבל את הטבלה הבאה:

```
0    0              charles
1    s(0)           son(charles)
2    s(s(0))        son(son(charles))
3    s(s(s(0)))     son(son(son(charles)))
```

וכן הלאה. המספר n מתבטא באמצעות הופעה n פעמית של s.

איור 3 – מספרים טבעיים המיוצגים כצאצאיו של 0

התכנית הבאה מגדירה את המושג של מספר טבעי באמצעות תכנית לוגית. כמו כן, היא מהווה בסיס להגדרתן של פעולות החשבון בדרך לוגית. את ההוכחות המתמטיות להגדרות אלו ניתן למצוא בפרק 3 בספר [Sterling & Shapiro, 1986].

```
natural_number(0).
natural_number(s(X)) :-
        natural_number(X).
```

תכנית 4(ב) – הגדרה של מספר טבעי

```
tracing: ¦ ?- natural_number(X).
[0] call: natural_number(X80)?
[0] exit: natural_number(0)?
... in 1 reductions.

X = 0 ;
[0] redo: natural_number(0)?
 [1] call: natural_number(X172)?
 [1] exit: natural_number(0)?
[0] exit: natural_number(s(0))?
... in 2 reductions.

X = s(0) ;
[0] redo: natural_number(s(0))?
 [1] redo: natural_number(0)?
  [2] call: natural_number(X254)?
  [2] exit: natural_number(0)?
 [1] exit: natural_number(s(0))?
[0] exit: natural_number(s(s(0)))?
... in 3 reductions.

X = s(s(0))
```

```
tracing: ¦ ?- clever(X).
[0] call: clever(X80)?
[0] exit: clever(charles)?
... in 1 reductions.

X = charles ;
[0] redo: clever(charles)?
 [1] call: clever(X172)?
 [1] exit: clever(charles)?
[0] exit: clever(son(charles))?
... in 2 reductions.

X = son(charles) ;
[0] redo: clever(son(charles))?
 [1] redo: clever(charles)?
  [2] call: clever(X254)?
  [2] exit: clever(charles)?
 [1] exit: clever(son(charles))?
[0] exit: clever(son(son(charles)))?
... in 3 reductions.

X = son(son(charles))
```

איור 4 – חישוב צאצאיו של צ'רלס ושל 0

נחזור למיועדנו צ'רלס. מהתכנית 1 ניתן להסיק, כי הוא היה איש מלא חיים, שאהב לבלות. ולעומתו בנו היה איש שקט, שאהב להיות בביתו עם בני משפחתו. נכדו היה איש מלא חיים שאהב לבלות, נינו היה איש שקט וחוזר חלילה. תכונות אופי אלו של צ'רלס וצאצאיו ניתן לבטא באמצעות התכנית הלוגית הבאה:

```
character(charles, society_man).
character(son(charles), family_man).
character(son(son(X)), Y) :-
                character(X, Y).
```

תכנית 4(ג) – תכונות אופי של צ׳רלס וצאצאיו

התכנית שלהלן מסווגת את המספרים הטבעיים לזוגיים ואי-זוגיים. השווה נא בינה לבין התכנית 4(ג), אשר באמצעותה תיארנו את תכונות האופי של צ׳רלס וצאצאיו.

```
number(0, even).
number(s(0), odd).
number(s(s(X)), Y) :-
                number(X, Y).
```

תכנית 4(ד) – מספרים זוגיים ואי-זוגיים

בסדרת תכניות אשר להלן, נגדיר פעולות שונות בין המספרים הטבעיים באמצעות חוקים רקורסיביים.

ה

```
/* A - less_or_equal(A, B) איננו גדול מ-B */

less_or_equal(0, X) :- natural_number(X).
less_or_equal(s(X), s(Y)) :- less_or_equal(X, Y).
```

ו

```
/* C - plus(A, B, C) הינו הסכום של A ו-B */

plus(0, X, X) :- natural_number(X).
plus(s(X), Y, s(Z)) :- plus(X, Y, Z).
```

ז

```
/* C - times(A, B, C) הינה מכפלה של A ו-B */

times(0, X, 0) :- natural_number(X).
times(s(X), Y, Z) :- times(X, Y, Q), plus(Q, Y, Z).
```

ח

```
/* F - factorial(N, F) הינו העצרת של N */

factorial(0, s(0)).
factorial(s(N), F) :- factorial(N, F1), times(s(N), F1, F).
```

תכניות 4(ה-ח) – פעולות במספרים טבעיים בדרך לוגית

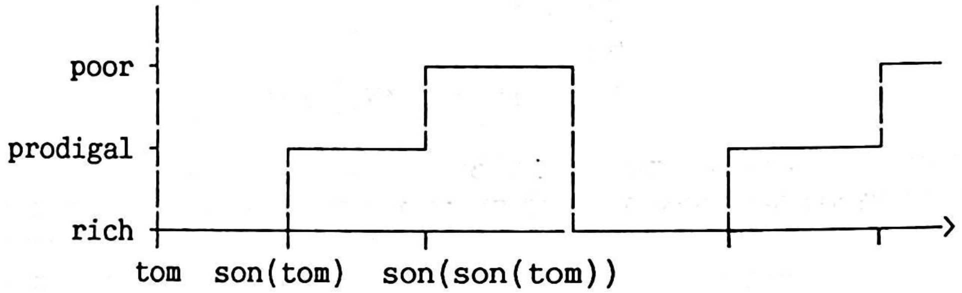
הערה: את מושג העצרת נגדיר בסעיף הבא.

נסיים את הסעיף הזה בתכנית, אשר מיישמת את הפתגם הבא: הדור הראשון בונה את העושר, הדור השני מבזבז, הדור השלישי מתרושש, הדור הרביעי בונה, וחוזר חלילה.

```
generation(tom, rich).
generation(son(tom), prodigal).
generation(son(son(tom)), poor).
generation(son(son(son(A))), X) :- generation(A, X).
```

תכנית 4(ט) – בונה-מבזבז-מתרושש

את תכנית 4(ט) אפשר לתאר באמצעות גרף, אשר ציר ה-X שלו הוא ציר הדורות וציר ה-Y שלו הוא ציר העושר.



איור 5 – הקשר בין הדורות לבין העושר

תרגיל 4

א. מצא את ארבע התשובות הראשונות לכל אחת מהשאילתות האלו:

```
?- less_or_equal(A, B).
?- plus(A, B, C).
?- times(A, B, C).
?- factorial(N, F).
```

תאר את כל אחת מהתשובות באמצעות העץ.

ב. מצא את שלוש התשובות הראשונות לכל אחת מהשאילתות האלו:

```
?- generation(X, rich).
?- generation(X, Y).
```

תאר את כל אחת באמצעות עץ.

ג. האם, לפי מיטב הבנתך, ניתן לקבל את הביטוי הבא:

```
less_or_equal(s(0), s(s(0)))
```

בתשובה לשאילתה:

```
less_or_equal(A, B)
```

נמק את תשובתך, ובדוק אותה באמצעות גירסת הפרולוג אשר ברשותך.

5. חישוב עצרת של מספר

חומר רקע: פרקים 5, 7, 8.

פונקצית העצרת f(N) מוגדרת עבור כל מספר טבעי N באופן הבא:

$$f(N) = \begin{cases} 1 & N=0 \\ 1 & N=1 \\ 1*2*...*N & N>1 \end{cases}$$

הסימון המקובל של הפונקציה הזאת הינו !N, אשר על כן:

```
0! = 1
1! = 1
N! = 1*2*...*N,  N>1
```

בסעיף הקודם הצגנו תכנית לוגית 4(ח) לחישוב העצרת. המספרים הטבעיים הוצגו בתכנית הזאת בתור צאצאיו של 0. בסעיף זה נבנה תכניות לחישוב העצרת, אשר בהן המספרים יוצגו בדרך המקובלת: 1, 2, וכן הלאה. את פעולות החשבון בתכניות אלו נבצע באמצעות פרדיקטי המערכת שלמדנו בפרק 8.

התכנית 5(א) מיישמת את הגדרת העצרת באמצעות חוק רקורסיבי, באופן דומה לזה של התכנית 4(ח): כדי לחשב און !N, חשב את !(N-1), והכפל את התוצאה ב-N. אין היא מפרטת איך לחשב את !(N-1). למעשה, החישוב של !(N-1) נעשה על פי אותו חוק, שניסחנו קודם: כדי לחשב את !(N-1), חשב את !(N-2), והכפל את התוצאה ב-(N-1). החוק הזה, אשר באמצעותו עוברים מ-!N ל-!(N-1) וכן הלאה, הינו חוק רקורסיבי. התנאי הגבולי מתבטא על ידי העובדה הבאה: העצרת של 0 שווה ל-1. להלן התכנית.

```
factorial1(N, F) :-
                  N>0,
                  N1 is N-1,
                  factorial1(N1, F1),
                  F is N*F1.
factorial1(0, 1).
```

תכנית 5(א) – חישוב העצרת

בהשוואה לתכנית 4(ח), יש בתכנית 5(א) נסיגה מן העקרון של תכנות לוגי. הנסיגה הזאת מוצאת את ביטויה בהודעת שגיאה, אשר מקבלים בתגובה לשאילתה factorial1(N, F). הודעת השגיאה נשלחת משום שההשוואה בין N לבין 0 מתבצעת רק כאשר תחת המשתנה N הוצב מספר. לעומת זה, כאשר בסעיף הקודם הצגנו את השאילתה factorial(N, F), זכינו לשלל של תשובות.

התכנית 5(א) אינה יעילה. היא איטית למדי, וגם צורכת נפח רב בזכרון (בעת חישוב של !N, מחסנית הזכרון גדלה עד לסדר גודל של N). נציין גם, כי לפרולוג אין אמצעים להחסנה של תוצאות הביניים (להוציא את פרדיקט המערכת assert, אשר הכרנו בפרק 12; כזכור, השימוש

בו בעייתי עקב התכונות האקסטרה-לוגיות שלו). יוצא אם כן, כי כדי לחשב את !5, מחשבים את !4, את !3, וכן הלאה, וכאשר רוצים לחשב את !6, חייבים לשוב ולחשב את !5, !4, וחוזר חלילה.

בתכנית 5(ב) נציג את שיטת הצבירה האיטרטיבית: נוסיף משתנה, שתפקידו יהיה לצבור את תוצאות הביניים. המשתנה הזה איננו כתובת בזכרון, כי אם משתנה לוגי. הצבירה תתבצע על ידי האיחוד בין תוצאת הביניים לבין המשתנה.
להלן התכנית. המשתנה האמצעי של factorial/3 הינו "הצובר".

```
/* N הינו העצרת של F - iterative_factorial(N, F) */

iterative_factorial(N, F) :- factorial(N, 1, F).
factorial(N, A, F) :-
                N>0,
                A1 is A*N,
                N1 is N-1,
                factorial(N1, A1, F).
factorial(0, F, F).
```

תכנית 5(ב) – חישוב עצרת בשיטת צבירה איטרטיבית

נציג תכנית איטרטיבית נוספת לחישוב העצרת. הפרוצדורה iterative_f/2 קוראת לפרוצדורה factorial/4. בפרוצדורה החדשה המשתנה הראשון מציין את לולאת האיטרציה (אשר דומה להוראת בייסיק: FOR I=0 TO N STEP 1; ניתן גם להשוותה עם הוראת while...do של פסקל).

```
iterative_f(N, F) :- factorial(0, N, 1, F).
factorial(I, N, A, F) :-
                I<N,
                J is I+1,
                A1 is A*J,
                factorial(J, N, A1, F).
factorial(N, N, F, F).
```

תכנית 5(ג) – עצרת איטרטיבית עם הלולאה

תרגיל 5

א. הצג את השאילתות שלהלן והשווה את התשובות (הפרוצדורות factorial ו-natural_number ניתנות בפרק 4 של חטיבה זו).

```
?- factorial(0, F).            ?- factorial1(0, F).
?- factorial(X, s(s(0))).      ?- factorial1(X, 2).
?- factorial(X, F).            ?- factorial1(X, F).
```

ב. חשב את !3 באמצעות כל אחת מהתכניות בסעיף זה. עקוב אחר החישוב בעזרת trace.

```
tracing: ¦ ?- iterative_factorial(6,F).
[0] call: iterative_factorial(6,F81)?
 [1] call: factorial(6,1,F173)?
  [2] call: 6 > 0?
  [2] exit: 6 > 0?
  [2] call: A1259 is 1 * 6?
  [2] exit: 6 is 1 * 6?
  [2] call: N1261 is 6 - 1?
  [2] exit: 5 is 6 - 1?
  [2] call: factorial(5,6,F257)?
   [3] call: 5 > 0?
   [3] exit: 5 > 0?
   [3] call: A1705 is 6 * 5?
   [3] exit: 30 is 6 * 5?
   [3] call: N1707 is 5 - 1?
   [3] exit: 4 is 5 - 1?
   [3] call: factorial(4,30,F703)?
    [4] call: 4 > 0?
    [4] exit: 4 > 0?
    [4] call: A11151 is 30 * 4?
    [4] exit: 120 is 30 * 4?
    [4] call: N11153 is 4 - 1?
    [4] exit: 3 is 4 - 1?
    [4] call: factorial(3,120,F1149)?
     [5] call: 3 > 0?
     [5] exit: 3 > 0?
     [5] call: A11597 is 120 * 3?
     [5] exit: 360 is 120 * 3?
     [5] call: N11599 is 3 - 1?
     [5] exit: 2 is 3 - 1?
     [5] call: factorial(2,360,F1595)?
      [6] call: 2 > 0?
      [6] exit: 2 > 0?
      [6] call: A12043 is 360 * 2?
      [6] exit: 720 is 360 * 2?
      [6] call: N12045 is 2 - 1?
      [6] exit: 1 is 2 - 1?
      [6] call: factorial(1,720,F2041)?
       [7] call: 1 > 0?
       [7] exit: 1 > 0?
       [7] call: A12489 is 720 * 1?
       [7] exit: 720 is 720 * 1?
       [7] call: N12491 is 1 - 1?
       [7] exit: 0 is 1 - 1?
       [7] call: factorial(0,720,F2487)?
        [8] call: 0 > 0?
        [8] fail: 0 > 0?
       [7] exit: factorial(0,720,720)?
      [6] exit: factorial(1,720,720)?
     [5] exit: factorial(2,360,720)?
    [4] exit: factorial(3,120,720)?
   [3] exit: factorial(4,30,720)?
  [2] exit: factorial(5,6,720)?
 [1] exit: factorial(6,1,720)?
[0] exit: iterative_factorial(6,720)?
... in 8 reductions.

F = 720
```

```
?- iterative_f(6,F).
iterative_f(6,F81)?
 factorial(0,6,1,F173)?
  0 < 6?
  0 < 6?
  J260 is 0 + 1?
  1 is 0 + 1?
  A1263 is 1 * 1?
  1 is 1 * 1?
  factorial(1,6,1,F259)?
   1 < 6?
   1 < 6?
   J706 is 1 + 1?
   2 is 1 + 1?
   A1709 is 1 * 2?
   2 is 1 * 2?
   factorial(2,6,2,F705)?
    2 < 6?
    2 < 6?
    J1152 is 2 + 1?
    3 is 2 + 1?
    A11155 is 2 * 3?
    6 is 2 * 3?
    factorial(3,6,6,F1151)?
     3 < 6?
     3 < 6?
     J1598 is 3 + 1?
     4 is 3 + 1?
     A11601 is 6 * 4?
     24 is 6 * 4?
     factorial(4,6,24,F1597)?
      4 < 6?
      4 < 6?
      J2044 is 4 + 1?
      5 is 4 + 1?
      A12047 is 24 * 5?
      120 is 24 * 5?
      factorial(5,6,120,F2043)?
       5 < 6?
       5 < 6?
       J2490 is 5 + 1?
       6 is 5 + 1?
       A12493 is 120 * 6?
       720 is 120 * 6?
       factorial(6,6,720,F2489)?
        6 < 6?
        6 < 6?
       factorial(6,6,720,720)?
      factorial(5,6,120,720)?
     factorial(4,6,24,720)?
    factorial(3,6,6,720)?
   factorial(2,6,2,720)?
  factorial(1,6,1,720)?
 factorial(0,6,1,720)?
iterative_f(6,720)?
in 8 reductions.

F = 720
```

איור 6 – חישוב של !6 באמצעות התכניות 5(ב), 5(ג)

6. פתרון בעיה באמצעות תכנות לוגי – מפת היעדים

חומר רקע: פרקים 5, 7, 8 (בלי פעולות החשבון).

אדם נודד מעיר לעיר בחיפוש אחר תעסוקה. את מצבו בעיר מסוימת הוא מגדיר כבעל סיכוי, אם בעיר זו יש תעסוקה, או אם ניתן להגיע מהעיר הזו לעיר אחרת שבה יש תעסוקה. כדי לעזור לאיש הנודד, החלטנו באמצעות תכנית לוגית לאתר את הערים אשר בהן מצבו הוגדר כבעל סיכוי. עיר, אשר בה מצבו הינו בעל סיכוי, תיקרא בשם "יעד". את התכנית נבנה בשיטה "מלמעלה למטה" (top-down). נגדיר תחילה "יעד":

יעד(_עיר) אם יש_סיכוי.

```
destination(Town) if has_chance.
```

את הסיכוי נסווג בהתאם למצב:

יש_סיכוי(_מצב).

```
has_chance(Situation).
```

מצב הינו נתון מבני:

מצב(_עיר, _תעסוקה).

```
situation(Town, Employment).
```

המצב הוא בעל-סיכוי, אם בעיר יש תעסוקה:

יש_סיכוי(מצב(_עיר, יש)) :-_עיר=ב; _עיר=ג; _עיר=ו.

```
has_chance (situation(Town, has)) :-Town=b; Town=c; Town=f.
```

המצב הוא בעל-סיכוי, אם ממנו ניתן להגיע אל מצב בעל-סיכוי:

יש_סיכוי(_מצב1):-מעבר(_מצב1, _מצב2), יש_סיכוי(_מצב2).

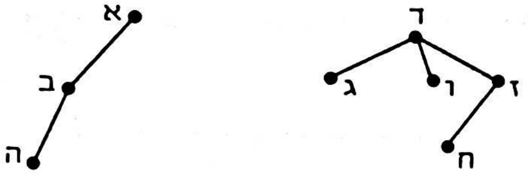
```
has_chance(Situation1):-move(Situation1, Situation2),
                        has_chance(Situation2).
```

נתנה את המעבר בשימוש בתחבורה העירונית ונתעלם מסוג ועלות התחבורה.

מעבר(מצב(_עיר1, _תעסוקה1), מצב(_עיר2,_תעסוקה2)):-
תחבורה(_עיר1, עיר2).

```
move(situation(Town1, Employ1), situation(Town2, Employ2)):-
                        communication(Town1, Town2).
```

להלן התכנית בשלמותה. את הערים נסמן באותיות עבריות בגירסה העברית של התכנית ובאותיות אנגליות קטנות – בגירסה האנגלית. המלה com היא קיצור של communication.




```
תחבורה(א, ב).
תחבורה(ב, ה).
תחבורה(ד, ג).
תחבורה(ד, ו).
תחבורה(ד, ז).
תחבורה(ז, ח).
מצב(א, אין).      /* בעיר א' אין תעסוקה */
מצב(ב, יש).       /* בעיר ב' יש תעסוקה */
מצב(ה, אין).
מצב(ח, אין).
מצב(ז, אין).
מצב(ד, אין).
מצב(ו, יש).
מצב(ג, יש).
מעבר(מצב(_עיר1, _תעסוקה1), מצב(_עיר2, _תעסוקה2)) :-
        תחבורה(_עיר1, _עיר2).

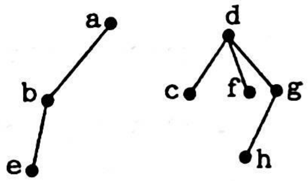
יש_סיכוי(מצב(_עיר, יש)) :- _עיר=ב; _עיר=ג; _עיר=ו.
יש_סיכוי(_מצב) :- מעבר(_מצב1, _מצב2), יש_סיכוי(_מצב2).

יעד(_עיר) :- יש_סיכוי(מצב(_עיר, _תעסוקה)).
```

תכנית 6(א) - מפת היעדים (גירסה עברית)

```
com(a, b).      /* com means communication */
com(b, e).
com(d, c).
com(d, f).
com(d, g).
com(g, h).
situation(a, has_not).
situation(b, has).
situation(e, has_not).
situation(h, has_not).
situation(g, has_not).
situation(d, has_not).
situation(f, has).
situation(c, has).
move(situation(Town1, Employ1), situation(Town2, Employ2)) :-
                                        com(Town1, Town2).

has_chance(situation(Town,has)) :-
                          Town = b; Town = c; Town = f.
has_chance(Situation1):- move(Situation1, Situation2),
                          has_chance(Situation2).
destination(Town) :- has_chance(situation(Town,Employment)).
```



תכנית 6(ב) – מפת היעדים (גירסה אנגלית)

התכנית הלוגית "מפת היעדים" מיישמת כמה נושאים חשובים. הנושא האחד הינו רקורסיה. ההגדרה "יש_סיכוי/1" היא הגדרה רקורסיבית. הנושא האחר הינו "נתון מבני", אשר דנו בו בפרק 8.

תרגיל 6

א. הצג כל אחד מהביטויים שלהלן באמצעות מבנה עץ.

```
1. has_chance(Situation)
2. has_chance(situation(Town, Employment))
3. move(Situation1, Situation2)
4. move(situation(Town1,Emp1), situation(Town2,Emp2))
```

ב. מצא את כל התשובות לשאילתות הבאות:

```
?- move(X, Y).
?- move(situation(X1, Y1), situation(X2, Y2)).
?- has_chance(X).
?- has_chance(situation(X, Y)).
```

ג. מצא את כל התשובות לשאילתה ?-destination(Town). אחת מהתשובות מופיעה פעמיים. איזו ומדוע? נמק את תשובתך.

ד. ברצוננו לתרגם לשאילתה של פרולוג את השאלה הבאה:

לאיזה ערים ניתן להגיע מהעיר, אשר בה יש תעסוקה?

נתבונן בשאילתה:

```
?- move(situation(Town, has), Situation).
```

האם השאילתה הזאת אכן מבטאת את השאלה שלעיל? נמק את תשובתך.

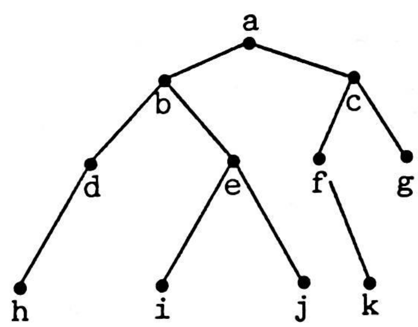
7. מודל לפתרון בעיה באמצעות תכנות לוגי

חומר רקע: פרקים 5, 7, 8.

ניתן לראות את התהליך של פתרון הבעיה כמורכב משני הצעדים האלה:
א. ייצוג הבעיה.
ב. קביעת אסטרטגיה של חיפוש התשובה.

בסעיף זה נבנה מודל לפתרון הבעיה באמצעות תכנית לוגית. במודל זה ניישם כמה מהשיטות והטכניקות של בינה מלאכותית. הבעיה בתחום של בינה מלאכותית מוצגת באמצעות **מרחב המצבים**. את המושג הזה היכרנו בהקדמה. מרחב המצבים הינו גרף, אשר צמתיו מייצגים את המצבים, והקווים המחברים את הצמתים מייצגים את המעברים ממצב למצב. לצורך בניית המודל נסמן את המצבים באותיות לועזיות מ-a ועד k (ראה איור 7). ממצב a ניתן לעבור למצב b או c. ממצב b ניתן לעבור למצב d או e, וכן הלאה. את המעברים נציג באופן פורמלי באמצעות היחס move, ובאופן בלתי פורמלי – באמצעות העץ שבאיור 7.

```
move(a,b).
move(a,c).
move(b,d).
move(b,e).
move(d,h).
move(e,i).
move(e,j).
move(c,f).
move(c,g).
move(f,k).
```

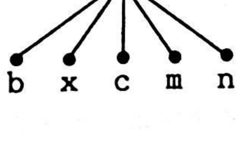


איור 7 – מרחב המצבים

את היחס ואת העץ נבנה בהתאם למוסכמות הבאות:

1. מהמצב a ניתן להגיע למצב b ולמצב c. המעברים האלו מיוצגים באמצעות היחסים move(a,b) ו-move(a,c). אי לכך, לצומת a של העץ, אשר באיור 7, ישנם שני בנים: b ו-c.
2. נקפיד על ההתאמה בין סדר כתיבת העובדות לבין סדר הופעת הבנים: אם העובדה move(a,b) נכתבה לפני העובדה move(a,c), אזי הבן b יופיע משמאלו של הבן c. באופן כללי נתאר את ההתאמה בין סדר כתיבת העובדות לבין סדר הופעת הבנים באמצעות איור 8:

```
move(a,b).
move(a,x).
move(a,m).
move(a,c).
move(a,n).
```



איור 8 – התאמה בין כתיבת העובדות לבין הופעת הבנים

את הבעיה ננסח במונחים של מרחב המצבים באופן הבא:

האם ניתן להגיע ממצב X למצב Y באמצעות המעברים הקיימים?

נפתור בעיה זו באמצעות תכנית לוגית, שמיישמת את השיקול הבא. ניתן להגיע ממצב X למצב Y, אם קיימת שרשרת מעברים: מ-X למצב ביניים Link, מ-Link למצב אחר וכך הלאה, עד שהמעבר האחרון של השרשרת יוביל אל המצב Y. את השיקול הזה נבטא באמצעות שני חוקים, אשר האחד מהם הינו חוק רקורסיבי, והאחר הינו התנאי הגבולי:

	כלל	חוק של פרולוג
1	ניתן להגיע מ-X ל-Y אם יש מעבר מ-X ל-Y	`solve1(X, Y) :-` `move(X, Y).`

2	ניתן להגיע מ-X ל-Y
	אם יש מעבר מ-X ל-Link
	ואם ניתן להגיע מ-Link ל-Y
		`solve1(X, Y) :-`
		`move(X, Link),`
		`solve1(Link, Y).`

תכנית 7(א) – החיפוש לעומק

אם נתבונן היטב בשני החוקים של היחס solve1, נמצא אותם מוכרים. באמצעות חוקים מסוג זה פתרנו בפרק 7 את בעיית אבות-אבותינו. בתור המצבים שימשו בני המשפחה, המעבר ממצב למצב התבטא באמצעות היחס father, ואת שרשרת האבות מצאנו בעזרת היחס ancestor. גם את בעיית הקשר הבין-עירוני, שהוצגה בפרק 7, פתרנו על פי אותו מודל. המעבר מ-X ל-Y התבצע באמצעות היחס road(X,Y) והפתרון התבטא באמצעות היחס connected(X,Y).

בבניית מודל solve1 נעזרנו באסטרטגיה לחיפוש הפתרון, הקרויה **חיפוש לעומק** (depth-first search). בחיפוש זה, מרחב המצבים נסרק באופן הבא: מצומת X עוברים באמצעות המעבר הקיים אל הבן הבכור. אם הבן הבכור של X הינו Y, מסיימים את החיפוש ואם הוא איננו Y- מסמנים אותו באמצעות משתנה, השונה מ-Y (למשל Link). עתה עוברים אל הבן הבכור של Link וכן הלאה. כשמגיעים לצומת, אשר איננו Y, ואשר אין לו בנים, מבצעים **נסיגה** (backtracking): חוזרים לאב של העלה ופונים אל הבן השני (נזכיר כי לצומת, אשר אין לו בנים קוראים עלה, ראה פרק 8). אם הבן השני איננו Y, פונים אל הבן הבכור של הבן השני, וכן הלאה. נדגים את החיפוש לעומק עם הנסיגה באמצעות ה-trace, שבאיור 9.

בנוסף לחיפוש לעומק, ניתן לפתור בעיות באמצעות **חיפוש לרוחב** (breadth-first search), או **חיפוש האוריסטי** (heuristic search). את התיאור האלגוריתמי של השיטות ניתן למצוא באחד מהספרים [Rich, 1986] ו-[Barr & Feigenbaum, 1982]. את היישום בפרולוג של איסטרטגיות החיפוש השונות ניתן למצוא בספר [Sterling & Shapiro, 1986].
כפי שראינו, המודל solve1 פותר את הבעיה באמצעות החיפוש לעומק. נציג גירסאות נוספות של המודל הזה. בגירסה solve2 משתמשים, כאשר המצב Y, שאליו רוצים להגיע, מוגדר בבסיס הנתונים כמצב סופי באמצעות היחס final_state. נניח שרוצים להגיע לאחד מהמצבים j או f. לשם כך, נוסיף לתכנית את שתי העובדות האלה:

```
final_state(j).
final_state(f).
```

היעזר באיור 7

```
tracing: | ?- solvel(a,k).
[0] call: solvel(a,k)?
 [1] call: move(a,k)?
 [1] fail: move(a,k)?
 [1] call: move(a,Link171)?
 [1] exit: move(a,b)?  <----  עבור מ-a לבנו הבכור b:
 [2] call: solvel(b,k)?
  [3] call: move(b,k)?
  [3] fail: move(b,k)?
  [3] call: move(b,Link389)?
  [3] exit: move(b,d)?  <----  עבור מ-b לבנו הבכור d:
  [4] call: solvel(d,k)?
   [5] call: move(d,k)?
   [5] fail: move(d,k)?
   [5] call: move(d,Link607)?
   [5] exit: move(d,h)?  <----  עבור מ-d לבנו הבכור h:
   [6] call: solvel(h,k)?
    [7] call: move(h,k)?
    [7] fail: move(h,k)?
    [7] call: move(h,Link825)?
    [7] fail: move(h,_825)?  <----  ל-h אין בנים:
   [6] fail: solvel(h,k)?
   [5] redo: move(d,h)?  <----  חזור מ-h ל-d:
   [5] fail: move(d,_607)?
  [4] fail: solvel(d,k)?
  [3] redo: move(b,d)?  <----  חזור מ-d ל-b:
  [3] exit: move(b,e)?  <----  עבור מ-b לבנו השני e:
  [4] call: solvel(e,k)?
   [5] call: move(e,k)?
   [5] fail: move(e,k)?
   [5] call: move(e,Link607)?
   [5] exit: move(e,i)?  <----  עבור מ-e לבנו הבכור i:
   [6] call: solvel(i,k)?
    [7] call: move(i,k)?
    [7] fail: move(i,k)?
    [7] call: move(i,Link825)?
    [7] fail: move(i,_825)?  <----  ל-i אין בנים:
   [6] fail: solvel(i,k)?
   [5] redo: move(e,i)?  <----  חזור מ-i ל-e:
   [5] exit: move(e,j)?
   [6] call: solvel(j,k)?
    [7] call: move(j,k)?
    [7] fail: move(j,k)?
    [7] call: move(j,Link825)?
    [7] fail: move(j,_825)?
   [6] fail: solvel(j,k)?
   [5] redo: move(e,j)?
   [5] fail: move(e,_607)?
  [4] fail: solvel(e,k)?
  [3] redo: move(b,e)?  <----  חזור מ-e ל-b:
  [3] fail: move(b,_389)?
 [2] fail: solvel(b,k)?
 [1] redo: move(a,b)?  <----  חזור מ-b ל-a:
 [1] exit: move(a,c)?  <----  עבור מ-a לבנו השני c:
 [2] call: solvel(c,k)?
  [3] call: move(c,k)?
  [3] fail: move(c,k)?
  [3] call: move(c,Link389)?
  [3] exit: move(c,f)?  <----  עבור מ-c לבנו הבכור f:
  [4] call: solvel(f,k)?
   [5] call: move(f,k)?
   [5] exit: move(f,k)?  <----  עבור מ-f לבנו הבכור k:
  [4] exit: solvel(f,k)?
 [2] exit: solvel(c,k)?
[0] exit: solvel(a,k)?
... in 6 reductions.
```

איור 9 – החיפוש לעומק באמצעות היחס solvel

להלן היחס solve2:

	כלל	חוק של פרולוג
1	המצב X הינו מצב סופי	solve2(X) :- final_state(X).
2	ניתן להגיע מ-X למצב הסופי אם יש מעבר מ-X ל-Link ואם ניתן להגיע מ-Link למצב הסופי	solve2(X) :- move(X,Link), solve2(Link).

תכנית 7(ב) – החיפוש לעומק, כאשר המצב הסופי מוגדר מראש

את היחס has_chance של תכנית "מפת היעדים" שבסעיף הקודם, הגדרנו על בסיס הגירסה solve2 של המודל.

נניח, כי נתון המצב a. האם ניתן להגיע ממנו אל המצב הסופי f או j? את השאלה הזאת נבטא בפרולוג באמצעות השאילתה ?-solve2(a).

תרגיל 7

א. תרגם את היחסים solve1 ו-solve2 לפרולוג העברי. היעזר במילון הבא:

פתור_בעיה	-	solve
מעבר	-	move
מצב_סופי	-	final_state

ב. היחסים אשר להלן, הופיעו בחלקן בפרק 7 של חטיבה שניה, וחלקן בחטיבה זו. נתח כל אחת מהן וסווג אותן על פי השתייכותן לגירסה מסוימת של מודל לפתרון הבעיה.

פתור_בעיה1 או פתור_בעיה2	solve1 או solve2
אב_בשושלת (סעיף 7.8 חטיבה שניה)	connected (סעיף 7.9)
מסקנה (סעיף 7.9 חטיבה שניה)	is_in (סעיף 7.9)
שרשרת (סעיף 2 חטיבה רביעית)	has_chance (סעיף 6)
עושה (סעיף 2 חטיבה רביעית)	
יש_סיכוי (סעיף 6 חטיבה רביעית)	

```
tracing: | ?- solve2(X).                          היעזר באיור 7
[0] call: solve2(X80)?
 [1] call: final_state(X171)?
 [1] exit: final_state(j)?
[0] exit: solve2(j)?
... in 2 reductions.

X = j ;
[0] redo: solve2(j)?
 [1] redo: final_state(j)?
 [1] exit: final_state(f)?
[0] exit: solve2(f)?
... in 2 reductions.

X = f ;
[0] redo: solve2(f)?
 [1] redo: final_state(f)?
 [1] fail: final_state(_171)?
 [1] call: move(X171,Link172)?
 [1] exit: move(a,b)?                             מעבר מ-a ל-b
 [2] call: solve2(b)?
  [3] call: final_state(b)?
  [3] fail: final_state(b)?
  [3] call: move(b,Link389)?
  [3] exit: move(b,d)?                            מעבר מ-b ל-d
  [4] call: solve2(d)?
   [5] call: final_state(d)?
   [5] fail: final_state(d)?
   [5] call: move(d,Link606)?
   [5] exit: move(d,h)?                           מעבר מ-d ל-h
   [6] call: solve2(h)?
    [7] call: final_state(h)?
    [7] fail: final_state(h)?
    [7] call: move(h,Link823)?
    [7] fail: move(h,_823)?
   [6] fail: solve2(h)?
   [5] redo: move(d,h)?                           נסיגה מ-h ל-d
   [5] fail: move(d,_606)?
  [4] fail: solve2(d)?                            נסיגה מ-d ל-b
  [3] redo: move(b,d)?
  [3] exit: move(b,e)?                            מעבר מ-b ל-e
  [4] call: solve2(e)?
   [5] call: final_state(e)?
   [5] fail: final_state(e)?
   [5] call: move(e,Link606)?
   [5] exit: move(e,i)?                           מעבר מ-e ל-i
   [6] call: solve2(i)?
    [7] call: final_state(i)?
    [7] fail: final_state(i)?
    [7] call: move(i,Link823)?
    [7] fail: move(i,_823)?
   [6] fail: solve2(i)?
   [5] redo: move(e,i)?                           נסיגה מ-i ל-e
   [5] exit: move(e,j)?                           מעבר מ-e ל-j
   [6] call: solve2(j)?
    [7] call: final_state(j)?
    [7] exit: final_state(j)?
   [6] exit: solve2(j)?
  [4] exit: solve2(e)?
 [2] exit: solve2(b)?                             המסקנה כי מ-a ניתן
[0] exit: solve2(a)?                              להגיע למצב הסופי
... in 8 reductions.
```

בהמשך נקבל את התשובות הבאות:

X = a
X = b
X = e
X = c

איור 10 – חיפוש לעומק באמצעות היחס solve2

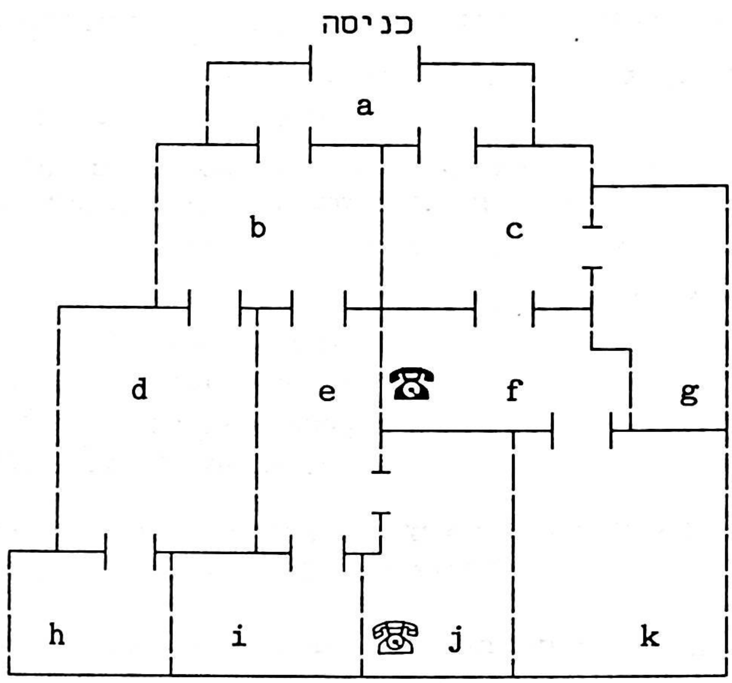
נציין, כי תכנית 7(ב) מאתרת תחילה את המצב j, אף-על-פי שהעומק של f הינו 2 והעומק של j הינו 3 (ראה איורים 7 ו-10).

8. חיפוש הדרך במבוך

חומר רקע: פרקים 5, 7, 8, 9.

נניח כי אנו נמצאים בארמון מפואר מן המאה ה-18. לאלה שלא נולדו בו, קשה להתמצא במבוך של החדרים, הפרוזדורים והמעברים. בשניים מחדריו של הארמון נמצאים מכשירי הטלפון אליהם אנו רוצים להגיע. נכתוב תכנית לוגית, אשר תציין את החדרים שיש לעבור, כדי להגיע למכשירי טלפון אלה.

תחילה נעיין באיור 12.11 המתאר את חדרי הארמון.



איור 12.11 – תרשים של הארמון

טלפון אחד נמצא בחדר f, והשני נמצא בחדר j. את החיפוש אחרי מכשיר הטלפון נתאר באמצעות היחס search. בעת החיפוש נקפיד לא לעבור פעמיים דרך אותו חדר, כדי לא להיקלע ללולאת המבוך. לכן, החיפוש יתבצע על פי הכללים האלה:

נניח כי אנו נמצאים בחדר X.

1. אם הטלפון נמצא בחדר X, אזי נסיים את החיפוש.
2. אם הטלפון אינו נמצא ב-X, אזי:
   - 2.1 נעבור לחדר אחר.
   - 2.2 נבדוק אם הוא שייך לרשימת החדרים שבהם כבר היינו.
   - 2.3 אם הוא אינו שייך לרשימה, נצרף אותו.
   - 2.4 נמשיך בחיפוש.

את רשימת החדרים שעברנו, נסמן באמצעות המשתנה Visited. בתחילת החיפוש יש ברשימה Visited איבר אחד בלבד – החדר שבו אנו נמצאים. כל חדר שדרכו נעבור, יצורף לרשימה זו.

בסיומו של החיפוש נקבל את רשימת החדרים, אשר דרכם יש לעבור כדי להגיע לטלפון. את הרשימה הזאת נסמן באמצעות המשתנה Rooms.

היחס search מתקיים בין שלושת העצמים האלה:

חדר X, אשר ממנו מתחיל החיפוש.

חדר Y, אשר אותו יש למצוא.

רשימה Rooms של חדרים, אשר דרכם עוברים מ-X ל-Y.

נעבור לכתיבת התכנית בפרולוג. בתחילת החיפוש יש ברשימה Visited איבר אחד בלבד.

```
search(X, Y, Rooms) :-
                search(X, Y, [X], Rooms).
```

אם הטלפון נמצא בחדר X, נסיים את החיפוש. במקרה זה הרשימה Visited תרכז את כל החדרים שדרכם עברנו כדי להגיע לטלפון, ועל כן תהיה זהה לרשימה Rooms:

```
search(X, X, Visited, Visited) :-
                        telephone(X).
```

אם הטלפון אינו נמצא בחדר X, נעבור לחדר Link ונבדוק אם הוא שייך לרשימה Visited. אם Link אינו שייך ל-Visited, נצרף את X לרשימה ונמשיך בחיפוש:

```
search(X, Y, Visited, Rooms) :-
                      move(X, Link),
                      not member(Link, Visited),
                      append(Visited, [Link], Visited1),
                      search(Link, Y, Visited1, Rooms).
```

נתאר את היחס move. הדלת בין שני חדרים סמוכים A ו-B מאפשרת את המעבר מ-A ל-B, וגם את המעבר מ-B ל-A. אשר-על-כן, היחס move מוגדר כך:

```
move(A, B) :- door(A, B); door(B, A).
```

נוסיף את המידע על הדלתות והטלפונים ונרכז לידו את החוקים שהגדרנו, כדי לקבל את התכנית בשלמותה:

```
door(a,b).
door(a,c).          move(A,B) :- door(A,B) ; door(B,A).
door(b,d).
door(b,e).          search(X, Y, Rooms) :-
door(d,h).                      search(X, Y, [X], Rooms).
door(e,i).          search(X, X, Visited, Visited) :-
door(e,j).                      telephone(X).
door(c,f).          search(X, Y, Visited, Rooms) :-
door(c,g).                      move(X, Link),
door(f,k).                      not member(Link, Visited),
telephone(f).                   append(Visited, [Link], Visited1),
telephone(j).                   search(Link, Y, Visited1, Rooms).
```

תכנית 8 – חיפוש דרך במבוך

הבה נכנס לארמון. אנו נמצאים בחדר a. איך נגיע לטלפון? נציג את השאילתה ונעיי
בתשובות:

```
?- search(a, Y, Rooms).

Y=j
Rooms = [a,b,e,j];

Y=f
Rooms = [a,c,f].
```

עקבנו אחרי התשובה לשאילתה הראשונה באמצעות ה-trace שבאיור 12. כדי לפשטו, הצגנ
את הפרוצדורה move באמצעות שני חוקים אלטרנטיביים:

```
move(A, B) :- door(A, B).
move(A, B) :- door(B, A).
```

התכנית שכתבנו מאפשרת להגיע לטלפון מכל אחד מחדרי הארמון:

```
?- search(h, Y, Rooms).

Y=j
Rooms = [h,d,b,e,j];

Y=f
Rooms = [h,d,b,a,c,f].
```

את התכנית בנינו בשלושה שלבים. בשלב הראשון דנו בבעיה ותיארנו באופן כללי את צעדי
הביצוע; בשלב השני גיבשנו את מבנה הנתונים; בשלב השלישי כתבנו את העובדות והחוקים
של פרולוג.

תרגיל 8

א. מצא את כל התשובות לשאילתות האלה:

```
?- search(X, f, Rooms).
?- search(X, j, Rooms).
```

ב. בתכנית 8 החלפנו את הפרוצדורה search בפרוצדורה search1:

```
search1(X, Y, Rooms) :-
                search1(X, Y, [], Rooms).
search1(X, X, Visited, Rooms) :-
                telephone(X),
                append(Visited, [X], Rooms).
search(X, Y, Visited, Rooms) :-
                move(X, Link),
                not member(Link, Visited),
                append(Visited, X, Visited1),
                search1(Link, Y, Visited1, Rooms).
```

תאר את ההבדל בין הפרוצדורות search ו-search1 ומצא, מה תהיינה התשובות לשאילתות האלו:

```
?- search1(a, Y, Rooms).
?- search1(h, Y, Rooms).
?- search1(X, f, Rooms).
?- search1(X, j, Rooms).
```

```
tracing: ¦ ?- search(a,Y,Rooms).
[0] call: search(a,Y81,Rooms82)?u
 [1] call: search(a,Y185,[a],Rooms186).
  [2] call: telephone(a).
  [2] fail: telephone(a).
  [2] call: move(a,Link268).
   [3] call: door(a,B362).
   [3] exit: door(a,b).
  [2] exit: move(a,b).
  [4] call: member(b,[a]).
  [4] fail: member(b,[a]).
  [4] call: append([a],[b],Visited1272).
  [4] exit: append([a],[b],[a,b]).
  [4] call: search(b,Y265,[a,b],Rooms267).
   [5] call: telephone(b).
   [5] fail: telephone(b).
   [5] call: move(b,Link721).
    [6] call: door(b,B815).
    [6] exit: door(b,d).
   [5] exit: move(b,d).
   [7] call: member(d,[a,b]).
   [7] fail: member(d,[a,b]).
   [7] call: append([a,b],[d],Visited1725).
   [7] exit: append([a,b],[d],[a,b,d]).
   [7] call: search(d,Y718,[a,b,d],Rooms720).
    [8] call: telephone(d).
    [8] fail: telephone(d).
    [8] call: move(d,Link1180).
     [9] call: door(d,B1274).
     [9] exit: door(d,h).
    [8] exit: move(d,h).
    [10] call: member(h,[a,b,d]).
    [10] fail: member(h,[a,b,d]).
    [10] call: append([a,b,d],[h],Visited11184).
    [10] exit: append([a,b,d],[h],[a,b,d,h]).
    [10] call: search(h,Y1177,[a,b,d,h],Rooms1179).
     [11] call: telephone(h).
     [11] fail: telephone(h).
     [11] call: move(h,Link1645).
      [12] call: door(h,B1739).
      [12] fail: door(h,_1739).
      [12] call: door(B1739,h).
      [12] exit: door(d,h).
     [11] exit: move(h,d).
     [13] call: member(d,[a,b,d,h]).
     [13] exit: member(d,[a,b,d,h]).
     [11] redo: move(h,d).
      [12] redo: door(d,h).
      [12] fail: door(_1739,h).
     [11] fail: move(h,_1645).
    [10] fail: search(h,_1177,[a,b,d,h],_1179).
    [10] redo: append([a,b,d],[h],[a,b,d,h]).
    [10] fail: append([a,b,d],[h],_1184).
```

א. עברנו את החדרים h, d, b, a.

ב. חזרנו לחדר d, גילנו כי היינו כאן.

בהמשך:

חזרנו לחדר b,

גילינו כי היינו כאן,

מחדר b הגענו לחדר i, חזרנו לחדר e, הגענו מ-e ל-j,

מצאנו טלפון, סיימנו את החיפוש.

```
     [11] call: telephone(j).
     [11] exit: telephone(j).
    [10] exit: search(j,j,[a,b,e,j],[a,b,e,j]).
   [7] exit: search(e,j,[a,b,e],[a,b,e,j]).
  [4] exit: search(b,j,[a,b],[a,b,e,j]).
 [1] exit: search(a,j,[a],[a,b,e,j]).
[0] exit: search(a,j,[a,b,e,j]).
... in 12 reductions.

Y = j
Rooms = [a,b,e,j]
```

איור 12 – חיפוש הטלפון

9. מודל לפתרון בעיה באמצעות תכנות לוגי (המשך)

חומר רקע: פרקים 5, 7, 8, 9, 10 (ידיעת פרק 10 רצוייה, אך אינה הכרחית).

בסעיף זה נמשיך לדון במודל לפתרון בעיה באמצעות תכנות לוגי. ברצוננו להרחיב את הפרוצדורה solve2 שבסעיף 7. ההרחבה אמורה לחשב את השביל בין המצב הנתון לבין המצב הסופי. במלים אחרות, אנו רוצים לרכז באמצעות הרשימה את כל המצבים, אשר דרכם עוברים כדי להגיע למצב הסופי. נגדיר, לשם כך את הפרוצדורה solve3:

```
solve3(X ,[X]) :-
        final_state(X).
solve3(X, [X|Path]) :-
        move(X, Link),
        solve3(Link, Path).
```

תכנית 10(א) – חיפוש אחר שביל

את המצבים, אשר עוברים בדרך אל המצב הסופי, נרכז באמצעות במשתנה Path.
נעיין בשאילתה ובתשובות שהתקבלו. את היחסים move ו-final_state הגדרנו בסעיף 7.

```
?- solve3(a, List).
List = [a,b,e,j];
List = [a,c,f].
```

נעקוב באמצעות איור 13 אחרי קבלת התשובה הראשונה. נשווה בין קבלת תשובה זו לבין קבלת התשובה לשאילתה solve2(a).

ישנן בעיות, אשר לא ניתן לפתור באמצעות הפרוצדורות solve2 או solve3. נניח למשל, כי המצבים מ-a ועד k מייצגים את הערים, המעברים מייצגים את הכבישים, והכביש המקשר את הערים b ו-d הינו דו-סיטרי (ראה איור 14). נוסיף לבסיס הנתונים של סעיף 7 את העובדה move(d,b) ונדגים באמצעות איור 15 את חישוב השאילתה ?-solve2(a).

מהמצב d ניתן להגיע אל שני המצבים h ו-b. את העובדה move(d,b) הוספנו לסוף הפרוצדורה move, ולכן כשהתכנית מגיעה אל המצב d, בהתאם למודל חישובי של תכנית לוגית, היא פונה אל המצב h. כאשר התכנית מגלה כי h אינו המצב הסופי, היא חוזרת אל d, ובאמצעות היחס move(d,b)- עוברת למצב b. מ-b היא עוברת ל-d, מ-d ל-h, וחוזר חלילה. וכך הפרוצדורה solve2(a) נכנסת ללולאה.

```
tracing: ¦ ?- solve2(a):
[0] call: solve2(a)?
 [1] call: final_state(a)?
 [1] fail: final_state(a)?
 [1] call: move(a,Link169)?
 [1] exit: move(a,b)?
 [2] call: solve2(b)?
  [3] call: final_state(b)?
  [3] fail: final_state(b)?
  [3] call: move(b,Link386)?
  [3] exit: move(b,d)?
  [4] call: solve2(d)?
   [5] call: final_state(d)?
   [5] fail: final_state(d)?
   [5] call: move(d,Link603)?
   [5] exit: move(d,h)?
   [6] call: solve2(h)?
    [7] call: final_state(h)?
    [7] fail: final_state(h)?
    [7] call: move(h,Link820)?
    [7] fail: move(h,_820)?
   [6] fail: solve2(h)?
   [5] redo: move(d,h)?
   [5] fail: move(d,_603)?
  [4] fail: solve2(d)?
  [3] redo: move(b,d)?
  [3] exit: move(b,e)?
  [4] call: solve2(e)?
   [5] call: final_state(e)?
   [5] fail: final_state(e)?
   [5] call: move(e,Link603)?
   [5] exit: move(e,i)?
   [6] call: solve2(i)?
    [7] call: final_state(i)?
    [7] fail: final_state(i)?
    [7] call: move(i,Link820)?
    [7] fail: move(i,_820)?
   [6] fail: solve2(i)?
   [5] redo: move(e,i)?
   [5] exit: move(e,j)?
   [6] call: solve2(j)?
    [7] call: final_state(j)?
    [7] exit: final_state(j)?
   [6] exit: solve2(j)?
  [4] exit: solve2(e)?
 [2] exit: solve2(b)?
[0] exit: solve2(a)?
... in 8 reductions.

yes
```

```
tracing: ¦ ?- solve3(a,List).
[0] call: solve3(a,List81)?
 [1] call: final_state(a)?
 [1] fail: final_state(a)?
 [1] call: move(a,Link175)?
 [1] exit: move(a,b)?
 [2] call: solve3(b,Path174)?
  [3] call: final_state(b)?
  [3] fail: final_state(b)?
  [3] call: move(b,Link394)?
  [3] exit: move(b,d)?
  [4] call: solve3(d,Path393)?
   [5] call: final_state(d)?
   [5] fail: final_state(d)?
   [5] call: move(d,Link613)?
   [5] exit: move(d,h)?
   [6] call: solve3(h,Path612)?
    [7] call: final_state(h)?
    [7] fail: final_state(h)?
    [7] call: move(h,Link832)?
    [7] fail: move(h,_832)?
   [6] fail: solve3(h,_612)?
   [5] redo: move(d,h)?
   [5] fail: move(d,_613)?
  [4] fail: solve3(d,_393)?
  [3] redo: move(b,d)?
  [3] exit: move(b,e)?
  [4] call: solve3(e,_393)?
   [5] call: final_state(e)?
   [5] fail: final_state(e)?
   [5] call: move(e,Link613)?
   [5] exit: move(e,i)?
   [6] call: solve3(i,Path612)?
    [7] call: final_state(i)?
    [7] fail: final_state(i)?
    [7] call: move(i,Link832)?
    [7] fail: move(i,_832)?
   [6] fail: solve3(i,_612)?
   [5] redo: move(e,i)?
   [5] exit: move(e,j)?
   [6] call: solve3(j,_612)?
    [7] call: final_state(j)?
    [7] exit: final_state(j)?
   [6] exit: solve3(j,[j])?
  [4] exit: solve3(e,[e,j])?
 [2] exit: solve3(b,[b,e,j])?
[0] exit: solve3(a,[a,b,e,j])?
... in 8 reductions.

List = [a,b,e,j]
```

איור 13 – קבלת התשובה ל-solve2(a) ול-solve3(a, List)

9. מודל לפתרון בעיה באמצעות תכנות לוגי (המשך)

חומר רקע: פרקים 5, 7, 8, 9, 10 (ידיעת פרק 10 רצוייה, אך אינה הכרחית).

בסעיף זה נמשיך לדון במודל לפתרון בעיה באמצעות תכנות לוגי. ברצוננו להרחיב את הפרוצדורה solve2 שבסעיף 7. ההרחבה אמורה לחשב את השביל בין המצב הנתון לבין המצב הסופי. במלים אחרות, אנו רוצים לרכז באמצעות הרשימה את כל המצבים, אשר דרכם עוברים כדי להגיע למצב הסופי. נגדיר, לשם כך את הפרוצדורה solve3:

```
solve3(X ,[X]) :-
            final_state(X).
solve3(X, [X|Path]) :-
            move(X, Link),
            solve3(Link, Path).
```

תכנית 10(א) – חיפוש אחר שביל

את המצבים, אשר עוברים בדרך אל המצב הסופי, נרכז באמצעות במשתנה Path.
נעיין בשאילתה ובתשובות שהתקבלו. את היחסים move ו-final_state הגדרנו בסעיף 7.

```
?- solve3(a, List).
List = [a,b,e,j];
List = [a,c,f].
```

נעקוב באמצעות איור 13 אחרי קבלת התשובה הראשונה. נשווה בין קבלת תשובה זו לבין קבלת התשובה לשאילתה solve2(a).

ישנן בעיות, אשר לא ניתן לפתור באמצעות הפרוצדורות solve2 או solve3. נניח למשל, כי המצבים מ-a ועד k מייצגים את הערים, המעברים מייצגים את הכבישים, והכביש המקשר את הערים b ו-d הינו דו-סיטרי (ראה איור 14). נוסיף לבסיס הנתונים של סעיף 7 את העובדה move(d,b) ונדגים באמצעות איור 15 את חישוב השאילתה ?-solve2(a).

מהמצב d ניתן להגיע אל שני המצבים h ו-b. את העובדה move(d,b) הוספנו לסוף הפרוצדורה move, ולכן כשהתכנית מגיעה אל המצב d, בהתאם למודל חישובי של תכנית לוגית, היא פונה אל המצב h. כאשר התכנית מגלה כי h אינו המצב הסופי, היא חוזרת אל d, ובאמצעות היחס move(d,b)- עוברת למצב b. מ-b היא עוברת ל-d, מ-d ל-h, וחוזר חלילה. וכך הפרוצדורה solve2(a) נכנסת ללולאה.

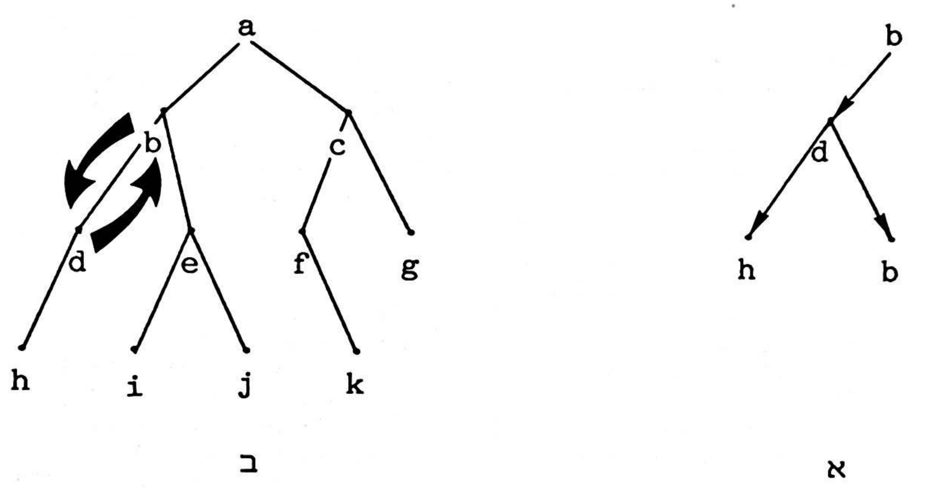
```
tracing: ¦ ?- solve2(a):
[0] call: solve2(a)?
 [1] call: final_state(a)?
 [1] fail: final_state(a)?
 [1] call: move(a,Link169)?
 [1] exit: move(a,b)?
 [2] call: solve2(b)?
  [3] call: final_state(b)?
  [3] fail: final_state(b)?
  [3] call: move(b,Link386)?
  [3] exit: move(b,d)?
  [4] call: solve2(d)?
   [5] call: final_state(d)?
   [5] fail: final_state(d)?
   [5] call: move(d,Link603)?
   [5] exit: move(d,h)?
   [6] call: solve2(h)?
    [7] call: final_state(h)?
    [7] fail: final_state(h)?
    [7] call: move(h,Link820)?
    [7] fail: move(h,_820)?
   [6] fail: solve2(h)?
   [5] redo: move(d,h)?
   [5] fail: move(d,_603)?
  [4] fail: solve2(d)?
  [3] redo: move(b,d)?
  [3] exit: move(b,e)?
  [4] call: solve2(e)?
   [5] call: final_state(e)?
   [5] fail: final_state(e)?
   [5] call: move(e,Link603)?
   [5] exit: move(e,i)?
   [6] call: solve2(i)?
    [7] call: final_state(i)?
    [7] fail: final_state(i)?
    [7] call: move(i,Link820)?
    [7] fail: move(i,_820)?
   [6] fail: solve2(i)?
   [5] redo: move(e,i)?
   [5] exit: move(e,j)?
   [6] call: solve2(j)?
    [7] call: final_state(j)?
    [7] exit: final_state(j)?
   [6] exit: solve2(j)?
  [4] exit: solve2(e)?
 [2] exit: solve2(b)?
[0] exit: solve2(a)?
... in 8 reductions.

yes
```

```
tracing: ¦ ?- solve3(a,List).
[0] call: solve3(a,List81)?
 [1] call: final_state(a)?
 [1] fail: final_state(a)?
 [1] call: move(a,Link175)?
 [1] exit: move(a,b)?
 [2] call: solve3(b,Path174)?
  [3] call: final_state(b)?
  [3] fail: final_state(b)?
  [3] call: move(b,Link394)?
  [3] exit: move(b,d)?
  [4] call: solve3(d,Path393)?
   [5] call: final_state(d)?
   [5] fail: final_state(d)?
   [5] call: move(d,Link613)?
   [5] exit: move(d,h)?
   [6] call: solve3(h,Path612)?
    [7] call: final_state(h)?
    [7] fail: final_state(h)?
    [7] call: move(h,Link832)?
    [7] fail: move(h,_832)?
   [6] fail: solve3(h,_612)?
   [5] redo: move(d,h)?
   [5] fail: move(d,_613)?
  [4] fail: solve3(d,_393)?
  [3] redo: move(b,d)?
  [3] exit: move(b,e)?
  [4] call: solve3(e,_393)?
   [5] call: final_state(e)?
   [5] fail: final_state(e)?
   [5] call: move(e,Link613)?
   [5] exit: move(e,i)?
   [6] call: solve3(i,Path612)?
    [7] call: final_state(i)?
    [7] fail: final_state(i)?
    [7] call: move(i,Link832)?
    [7] fail: move(i,_832)?
   [6] fail: solve3(i,_612)?
   [5] redo: move(e,i)?
   [5] exit: move(e,j)?
   [6] call: solve3(j,_612)?
    [7] call: final_state(j)?
    [7] exit: final_state(j)?
   [6] exit: solve3(j,[j])?
  [4] exit: solve3(e,[e,j])?
 [2] exit: solve3(b,[b,e,j])?
[0] exit: solve3(a,[a,b,e,j])?
... in 8 reductions.

List = [a,b,e,j]
```

איור 13 – קבלת התשובה ל-solve2(a) ול-solve3(a, List)



איור 14 – א. האב והצאצאים של d
ב. הלולאה

הפרוצדורה solve3(a) נכנסת גם כן ללולאה דומה. הגירסה הנוספת של המודל, solve4, נועדה למנוע את הכניסה ללולאה. הפרוצדורה solve4 צוברת באמצעות המשתנה History את המצבים שדרכם עברה התכנית. כל מצב נוסף נבדק באמצעות הפרוצדורה member, כדי לגלות אם התכנית עברה דרכו, אם לאו. להלן הפרוצדורה solve4:

```
solve4(X,Path) :- solve4(X,[],Path).
solve4(X,History,[X|History]) :- final_state(X).
solve4(X,History,Path) :- move(X,Link),
                          not member(Link,History),
                          solve4(Link,[X|History],Path).
```

תכנית 9(ב) – חיפוש שביל ללא לולאות

נציג את השאילתה ונעיין בתשובות:

```
?- solve4(a, List).
List = [j,e,b,a];
List = [f,c,a].
```

ניתן (בתוספת של append) לשנות את הפרוצדורה solve4 כך, שהרשימה List תופיעה בסדר הבא: [a,b,e,j], [a,c,f].

באיור 16 עקבנו אחרי חישוב התשובה הראשונה לשאילתה solve4(a, List). נעיין בחלק ב' של ה-trace, כאשר התכנית מגלה כי המצב b אכן שייך לרשימת המצבים שהצטברו ב-History.

```
| ?- trace.
tracing: | ?- solve2(a).
[0] call: solve2(a)?
 [1] call: final_state(a)?
 [1] fail: final_state(a)?
 [1] call: move(a,Link169)?
 [1] exit: move(a,b)?
 [2] call: solve2(b)?
  [3] call: final_state(b)?
  [3] fail: final_state(b)?
  [3] call: move(b,Link386)?
  [3] exit: move(b,d)?
  [4] call: solve2(d)?
   [5] call: final_state(d)?
   [5] fail: final_state(d)?
   [5] call: move(d,Link603)?
   [5] exit: move(d,h)?
   [6] call: solve2(h)?
    [7] call: final_state(h)?
    [7] fail: final_state(h)?
    [7] call: move(h,Link820)?
    [7] fail: move(h,_820)?
   [6] fail: solve2(h)?
   [5] redo: move(d,h)?
   [5] exit: move(d,b)?
   [6] call: solve2(b)?
    [7] call: final_state(b)?
    [7] fail: final_state(b)?
    [7] call: move(b,Link820)?
    [7] exit: move(b,d)?
    [8] call: solve2(d)?
     [9] call: final_state(d)?
     [9] fail: final_state(d)?
     [9] call: move(d,Link1037)?
     [9] exit: move(d,h)?
     [10] call: solve2(h)?
      [11] call: final_state(h)?
      [11] fail: final_state(h)?
      [11] call: move(h,Link1254)?
      [11] fail: move(h,_1254)?
     [10] fail: solve2(h)?
     [9] redo: move(d,h)?
     [9] exit: move(d,b)?
     [10] call: solve2(b)?
      [11] call: final_state(b)?
      [11] fail: final_state(b)?
      [11] call: move(b,Link1254)?
      [11] exit: move(b,d)?
      [12] call: solve2(d)?
       [13] call: final_state(d)?
       [13] fail: final_state(d)?
       [13] call: move(d,Link1471)?
       [13] exit: move(d,h)?
       [14] call: solve2(h)?
        [15] call: final_state(h)?
        [15] fail: final_state(h)?
```

איור 15 – הלולאה של solve2(a)

```
tracing: | ?- solve4(a,List).
[0] call: solve4(a,List81)?
 [1] call: solve4(a,[],Path173)?
  [2] call: final_state(a)?
  [2] fail: final_state(a)?
  [2] call: move(a,Link258)?
  [2] exit: move(a,b)?
  [3] call: member(b,[])?
  [3] fail: member(b,[])?
  [3] call: solve4(b,[a],Path257)?
   [4] call: final_state(b)?
   [4] fail: final_state(b)?
   [4] call: move(b,Link492)?
   [4] exit: move(b,d)?
   [5] call: member(d,[a])?
   [5] fail: member(d,[a])?
   [5] call: solve4(d,[b,a],Path491)?
    [6] call: final_state(d)?
    [6] fail: final_state(d)?
    [6] call: move(d,Link726)?
    [6] exit: move(d,h)?
    [7] call: member(h,[b,a])?
    [7] fail: member(h,[b,a])?
    [7] call: solve4(h,[d,b,a],Path725)?
     [8] call: final_state(h)?
     [8] fail: final_state(h)?
     [8] call: move(h,Link960)?
     [8] fail: move(h,_960)?
    [7] fail: solve4(h,[d,b,a],_725)?
    [6] redo: move(d,h)?
    [6] exit: move(d,b)?
    [7] call: member(b,[b,a])?
    [7] exit: member(b,[b,a])?
    [6] redo: move(d,b)?
    [6] fail: move(d,_726)?
   [5] fail: solve4(d,[b,a],_491)?
   [4] redo: move(b,d)?
   [4] exit: move(b,e)?
   [5] call: member(e,[a])?
   [5] fail: member(e,[a])?
   [5] call: solve4(e,[b,a],_491)?
    [6] call: final_state(e)?
    [6] fail: final_state(e)?
    [6] call: move(e,Link726)?
    [6] exit: move(e,i)?
    [7] call: member(i,[b,a])?
    [7] fail: member(i,[b,a])?
    [7] call: solve4(i,[e,b,a],Path725)?
     [8] call: final_state(i)?
     [8] fail: final_state(i)?
     [8] call: move(i,Link960)?
     [8] fail: move(i,_960)?
    [7] fail: solve4(i,[e,b,a],_725)?
    [6] redo: move(e,i)?
    [6] exit: move(e,j)?
    [7] call: member(j,[b,a])?
    [7] fail: member(j,[b,a])?
    [7] call: solve4(j,[e,b,a],_725)?
     [8] call: final_state(j)?
     [8] exit: final_state(j)?
    [7] exit: solve4(j,[e,b,a],[j,e,b,a])?
   [5] exit: solve4(e,[b,a],[j,e,b,a])?
  [3] exit: solve4(b,[a],[j,e,b,a])?
 [1] exit: solve4(a,[],[j,e,b,a])?
[0] exit: solve4(a,[j,e,b,a])?
... in 9 reductions.
List = [j,e,b,a]
```

איור 16 – תיקון הלולאה
באמצעות solve4(a, List)

בגירסאות 1 עד 4 של solve המעבר ממצב X למצב Y הוצג באמצעות היחס move(X,Y). לעיתים יש צורך לייצג את המעבר בצורה יותר מתוחכמת. נציג גירסה solve5 של המודל, אשר בה המעבר מוצג באמצעות היחס

```
move(Situation1, Move, Situation2)
```

נוסיף אל בסיס הנתונים את העובדות הבאות:

```
move(a, move(a,b) ,b).
move(a, move(a,c), c).
move(b, move(b,d), d).
move(b, move(b,e), e).
move(d, move(d,b), b).
move(e, move(e,j), j).
move(c, move(c,f), f).
```

להלן הפרוצדורה solve5:

```
solve5(X,Moves) :- solve5(X,[],Moves).
solve5(X,History,[]) :- final_state(X).
solve5(X,History,[Move|Moves]) :-  move(X,Move,Link),
                        not member(Link,History),
                        solve5(Link,[Link|History],Moves).
```

תכנית 9(ג) – חיפוש עם מעבר מתוחכם

נעיין שוב בפרוצדורה solve3. באמצעות החיפוש לעומק אין אנו מאתרים את הפתרון הקצר ביותר: התשובה [a,b,e,j] מופיעה לפני התשובה [a,c,f]. במקרים קיצוניים ניתן באמצעות החיפוש לעומק "להסתבך" בתוך הענף האינסופי של העץ, כפי שזה קורה לעתים בפרוצדורות רקורסיביות כדוגמת ancestor שבפרק 10. הגירסה solve6 נועדה להגביל את עומק עץ החיפוש. להלן הפרוצדורה solve6:

```
solve6(X,[X],_) :- final_state(X).
solve6(X,[X|Path],Maxdepth) :- Maxdepth > 0,
                  move(X,Link), Mxdpth is Maxdepth - 1,
                  solve6(Link,Path,Mxdpth).
```

תכנית 9(ד) – חיפוש עד עומק מסוים

נעיין בשאילתות ובתשובות שלהלן:

```
?- solve6(a, List, 1).
no
?- solve6(a, List, 2).
List = [a,c,f]
?- solve6(a, List, 3).
List = [a,b,e,j];
List = [a,c,f].
```

משמעות התשובות הינה כדלהלן: ברצוננו להגיע ממצב a למצב המוגדר כמצב סופי. בעומק 1 אין מצב סופי. בעומק אשר אינו עולה על 2, יש מצב סופי אחד בלבד, f, ואליו ניתן להגיע דרך המצב c. בעומק אשר אינו עולה על 3, ישנם שני מצבים סופיים, f ו-j.

נאמר לסיום, כי ניתן לבנות גירסאות נוספות של המודל. ניתן למשל, לתאר את המעבר ממצב למצב באמצעות שני יחסים:

```
move(State1, Move).
update(Move,State2).
```

נציין גם כי האסטרטגיה של החיפוש לעומק הינה חלק של מערכת הפרולוג, ולכן לא נזקקנו לתאור אלגוריתמי מורכב של המודל. התאור באמצעות הגירסאות שהגדרנו, היה שקוף והצהרתי.

את הפרוצדורות לפתרון הבעיה באמצעות תכנות לוגי רכזנו באיור 17.

```
solve1(X,Y) :- move(X,Y).                                              .1

solve1(X,Y) :- move(X,Link), solve1(Link,Y).

solve2(X) :- final_state(X).                                           .2
  solve2(X) :- move(X,Link), solve2(Link).

solve3(X,[X]) :- final_state(X).                                       .3
solve3(X,[X|Path]) :- move(X,Link), solve3(Link,Path).

solve4(X,Path) :- solve4(X,[],Path).                                   .4
solve4(X,History,[X|History]) :- final_state(X).
solve4(X,History,Path) :- move(X,Link),
                     not member(Link,History),
                     solve4(Link,[X|History],Path).

solve5(X,Moves) :- solve5(X,[],Moves).                                 .5
solve5(X,History,[]) :- final_state(X).
solve5(X,History,[Move|Moves]) :-  move(X,Move,Link),
                     not member(Link,History),
                     solve5(Link,[Link|History],Moves).

solve6(X,[X],_) :- final_state(X).                                     .6
solve6(X,[X|Path],Maxdepth) :- Maxdepth > 0,
                     move(X,Link), Mxdpth is Maxdepth - 1,
                     solve6(Link,Path,Mxdpth).
```

איור 17 – פתרון בעיה באמצעות תכנות לוגי

תרגיל 9

א. תאר את בעיית חיפוש הדרך במבוך (סעיף 8) בשפת מרחב המצבים וצייר את הגרף המתאים.

ב. נתח את הפרוצדורות search ו-search1 של סעיף 8 וסווג אותן על פי השתייכותן למודלים solve3, solve4, solve5, solve6.

ג. אם לא ניתן להתאים בין המודלים של solve לבין הפרוצדורה search (או search1), כתוב מודל חדש, אשר יתאים לפרוצדורה הנ"ל.



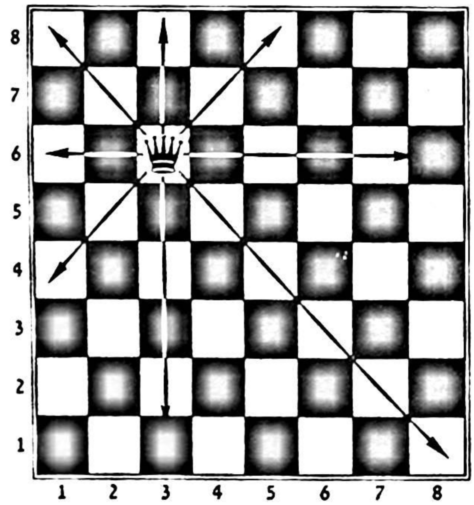
10. בעיית שמונה המלכות

חומר רקע: פרקים 5, 7, 8, 9.

האם ניתן למקם שמונה מלכות על גבי לוח שחמט ריק כך, שאף מלכה לא תאיים על רעותה? שאלה זו הוצגה לראשונה בכתב-עת גרמני לשחמט בשנת 1948. המלכה היא כלי של משחק שחמט, שיכולה לנוע מרחק כלשהו בכל אחד משמונת הכיוונים העיקריים שעל הלוח. המלכה נעה בכיוון השורה – ימינה. או שמאלה, בכיוון העמודה – מעלה או מטה, ובכל אחד מארבעת כיווני האלכסון (ראה איור 18). בשל כך מאיימת המלכה על כל כלי הנמצא באחד משמונת הכיוונים האפשריים.

נניח, כי מלכה אחת ניצבת במשבצת על פני לוח השחמט. על-כן בשורה בה נמצאת המלכה, לא תימצא כל מלכה אחרת. כלל זה חל גם על העמודה ועל ארבעת האלכסונים. האמנם נוכל להציב את כל שמונה המלכות על אותו לוח בלא שתאיימנה זו על זו? התשובה לכך חיובית. יתר על כן: הוכח, כי קיימות 92 אפשרויות למקם את המלכות על גבי הלוח. בכללן נמנות גם תמונות ראי במצבים שונים, ואם נפסול אותם, נגיע למספר אפשרויות קטן יותר.

נציין, כי בנושא זה נעשו מחקרים ונכתבו מאמרים רבים (ראה למשל, את המאמר [Schrage, 1986]).



איור 18 – המלכה נעה בשמונה כיוונים

את מלאכת הצבת המלכות על פני הלוח נעשה באמצעות תכנות לוגי. בעת כתיבת התכנית עלינו להשיב על שתי השאלות העיקריות:

א. כיצד יוצגו המלכות על הלוח?
ב. מה תהיה האסטרטגיה של חיפוש הפתרון?

נכתוב שלוש תכניות לוגיות, שבהן נציג גישות שונות לפתרון הבעיה. בתכנית הראשונה נתאר את כל אחת מהמלכות על פני הלוח באמצעות זוג סדור (X,Y) של שעורים (קואורדינטות). הערך של X יציין את העמודה והערך של Y את השורה, בה ניצבת המלכה. את תצורת המלכות (configuration of queens) נציג באמצעות הרשימה הבאה:

[(X1, Y1), (X2, Y2), ...(X8, Y8)]

כדי לפשט את ההצגה, נארגן את המלכות לפי הסדר הטבעי של העמודות:

[(1, Y1), (2, Y2), ...(8, Y8)]

את התכנית הראשונה נתחיל מהגדרת היחס queens1(Queens). המשתנה Queens של היחס מייצג את תצורת המלכות:

```
queens1(Queens):-
        Queens=[(1,Y1), (2,Y2), (3,Y3),
                (4,Y4), (5,Y5), (6,Y6), (7,Y7), (8,Y8)],
        solution1(Queens).
```

באמצעות היחס solution1(Queens) ניישם את האסטרטגיה לחיפוש הפתרון. כדי לקבל את התצורה, אשר בה אף לא מלכה אחת תאיים על רעותה, ניעזר בהגדרה רקורסיבית. נניח כי כמה מהמלכות כבר הוצבו על גבי הלוח בלא שאיימו זו על זו. נסמן מלכות אלו על ידי המשתנה Others. את המלכה הנוספת יש למקם על הלוח בלא שתאיים על Others. נקבל אם-כן, כי הצבת מספר כלשהו N של מלכות מושתת על שלב קודם, שבו הוצבו N-1 על פני הלוח:

```
solution1([(X, Y)|Others]):-
        solution1(Others),
        member(Y, [1, 2, 3, 4, 5, 6, 7, 8]),
        not attacks1((X, Y), Others).
```

"תנאי הגבול" הוא המקרה, שבו הלוח ריק:

```
solution([]).
```

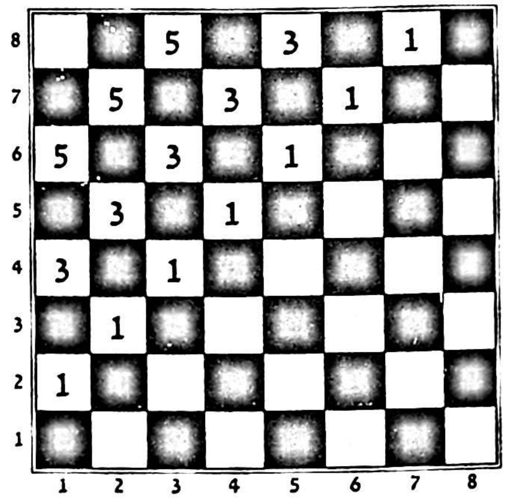
נעבור אל היחס attacks1(Queen,Others). המלכה תוקפת בכיוון העמודה:

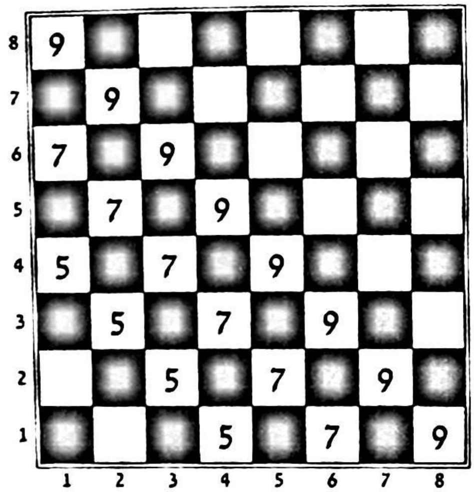
```
attacks1((X, Y), Others):-
        member((A, B) , Others),
        A = X.
```

המלכה תוקפת בכיוון השורה:

```
attacks1((X, Y), Others):-
        member((A, B) , Others),
        B = Y.
```

המלכה תוקפת בכיווני ארבעת האלכסונים. הבה נתבונן באלכסוני הלוח. אם נתבונן באלכסונים העולים משמאל לימין נגלה, כי ההפרש בין שיעורי המשבצות באלכסונים אלה הוא קבוע, האופייני לאלכסון (ראה איור 19). באלכסונים היורדים משמאל לימין סכום שיעורי המשבצות הוא קבוע, האופייני לאלכסון.





איור 19 – איפיונים של אלכסוני הלוח:
(א) הפרש בין שיעורים באלכסונים העולים משמאל לימין.
(ב) סכום של שיעורים באלכסונים היורדים משמאל לימין.

המלכה (X,Y) תוקפת את המלכה (A,B) בכיוון האלכסון שעולה משמאל לימין, אם: B-A=Y-X, או אם: B=Y-X+A. את השיקול הזה נבטא בפרולוג באמצעות החוק הבא:

```
attacks1((X, Y), Others):-
                  member((A, B), Others),
                  B  is  Y-X+A.
```

בדומה, המלכה (X,Y) תוקפת את המלכה (A,B) בכיוון האלכסון שיורד משמאל לימין, אם: B+A=Y+X, או אם: B=Y+X-A. החוק המתאים בפרולוג הוא:

```
attacks1((X, Y), Others):-
                  member((A, B), Others),
                  B  is  Y+X-A.
```

את היחס member(Member, List) הגדרנו בפרק 9.

לפני שנציג את התכנית הראשונה בשלמותה נציין, כי ניתן לוותר על החוק המגדיר התקפות בכיוון העמודה. מכיוון שהעמודות מסודרות לפי הסדר הטבעי, אין שתי מלכות הנמצאות באותה עמודה.

להלן התכנית Queens1 בשלמותה. את כתיבת חוקי attacks1 "חסכנו" על ידי שימוש בקשר ";" (OR). נזקקנו לסוגריים המקיפים את הפיסקאות הקשורות באמצעות הקשר OR, כדי לציין את סדר הקדימויות של הקשרים.

```
queens1(Queens):-
    Queens = [(1, Y1), (2, Y2), (3, Y3), (4, Y4),
              (5, Y5), (6, Y6), (7, Y7), (8, Y8)],
    solution1(Queens).

solution1([]).
solution1([(X, Y)|Others]):-
    solution1(Others),
    member(Y, [1, 2, 3, 4, 5, 6, 7, 8]),
    not attacks1((X, Y), Others).

attacks1((X,Y), Others):-
    member((A, B), Others),
    (B = Y ;  B  is  Y-X+A; B  is  Y+X-A).

member(Member, [Member|Rest]).
member(Member,[Head|Tail]):-
    member(Member, Tail).
```

תכנית 10(א) – פתרון בעית שמונה המלכות

קודם שנפנה לכתוב את התכנית השניה נעיר, כי אין כל סיבה שנתייחס רק לשמונה המלכות על פי הגודל המקובל של לוח שחמט. אפשר להכליל את הבעיה: האם ניתן למקם N מלכות על גבי לוח שגודלו N★N בלא שהמלכות יאיימו זו על זו?

אין תועלת רבה בשיעור ה-X של הזוג הסדור (X,Y). את המלכות ניתן להציג באמצעות רשימת שיעורי ה-Y. הערך של Y יציין את השורה בה נמצאת המלכה ומקומו של Y ברשימה יציין את העמודה.
כל מלכה חייבת לתפוס שורה משלה, כדי שלא תאיים על רעותה. לכן, רשימת שיעורי ה-Y היא תמורה (permutation) של המספרים מ-1 עד N.

את התכנית השניה נפתח בהגדרת היחס queens2(N, Queens). המשתנה Queens הוא תמורה המייצגת את תצורת המלכות:

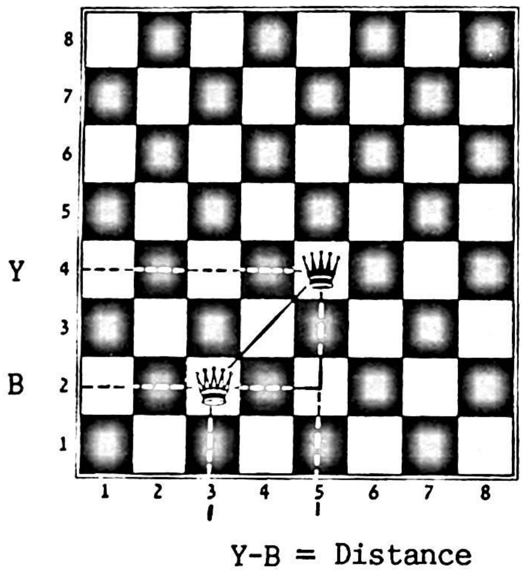
```
queens2(N, Queens) :-
      sequence(N, Sequence),
      permutation(Sequence, Queens),
      solution2(Queens).
```

היחס sequence(N, Sequence) מייצר את הרשימה Sequence של מספרים מ-1 עד N: [1,2,....,N].
היחס permutation(Sequence, Queens) מייצר את התמורה Queens של הרשימה Sequence,
והיחס solution2(Queens) בודק האם התמורה Queens מייצגת את תצורת המלכות בלא שהן יאיימו זו על זו. שיטת הפתרון, שמיושמת באמצעות היחסים permutation ו-solution2, מכונה בשם "ייצר-ובדוק" (generate-and-test). היחס permutation בונה את התצורות, והיחס solution2 בודק אם התצורה שהתקבלה אכן פותרת את הבעיה. נזכיר, כי באמצעות השיטה "ייצר-ובדוק" ביצענו גם מיון טבעי של רשימה (ראה פרק 9).

את היחס sequence ואת היחס permutation הגדרנו בפרק 9. עתה נגדיר את היחס solution2. הגדרתו תהיה רקורסיבית בדומה ל-solution1 של התכנית הראשונה:

```
solution2([]).
solution2([Y|Others]) :-
        solution2(Others),
        not attacks2(Y, Others).
```

את היחס attacks2 נגדיר על פי השיקולים הבאים: המלכה שנמצאת בשורה Y, מאיימת בכיוון האלכסון על המלכה שבשורה B, אם המרחק בין העמודות, שבהן נמצאות המלכות, שווה למרחק שבין השורות, כלומר, המרחק בין העמודות שווה ל-(Y-B) או ל-(B-Y) (ראה איור 20).



איור 20 – מלכות מאיימות

המלכות מאיימות זו על זו בכיוון האלכסון, אם המרחק בין השורות שווה למרחק בין העמודות. יוצא אם-כן, כי בהגדרת היחס attacks2 אנו נעזרים במרחק בין העמודות של המלכות בלא שמציינים במפורש את שיעורי העמודות. לשם כך אנו משנים את היחס attacks2 המקורי ועוברים ליחס attacks/3, אשר בו הארגומנט האחרון הינו המרחק בין העמודות:

```
attacks2(Y, Others):-
                attacks2(Y, Others, 1).
```

נניח, כי המרחק בין העמודות של המלכות שווה ל-Distance. על-פי השיקולים שצויינו לעיל, המלכות מאיימות זו על זו בכיוון האלכסון, אם המרחק בין השורות שלהן שווה גם הוא ל-Distance:

```
attacks2((Y, [B|Tail], Distance):-
                    Y  is  B + Distance;
                    Y  is  B - Distance.
```

האלכסונים נבדקים בזה אחר זה עד קצה הלוח:

```
attacks2(Y, [B|Tail], Distance):-
                  D1  is  Distance + 1,
                  attacks2(Y, Tail, D1).
```

להלן התכנית השניה בשלמותה:

```
queens2(N, Queens) :-
             sequence(N, Sequence),
             permutation(Sequence, Queens),
             solution2(Queens).
sequence(1, [1]).
sequence(N, Sequence):-
             N1  is  N-1,
             sequence(N1, S1),
             append(S1, [N], Sequence).
append([], L, L).
append([X|Tail], L, [X|NewTail]):-
              append(Tail, L, NewTail).
permutation(X, [Y|T]) :-
            select(Y,X,NewList),
            permutation(NewList,T).
permutation([],[]).
select(X,[X|T],T).
select(X,[Y|T],[Y|Z]):-
           select(X, T, Z).
solution2([]).
solution2([Y|Others]):-
            solution2(Others),
            not attacks2(Y, Others).
attacks2(Y, Queens) :-
            attacks2(Y, Queens, 1).
attacks2(Y, [B|Tail], Distance) :-
                      Y  is  B + Distance
                      Y  is  B - Distance
attacks2(Y, [B|Tail], Distance) :-
                      D1  is  Distance + 1,
                      attacks2(Y, Tail, D1).
```

תכנית 10(ב) – פתרון בעית המלכות באמצעות "ייצר-ובדוק"

בתכנית השניה מייצרים תמורה ובודקים, אם היא מייצגת את תצורת המלכות הרצויה. אם לא – שוב מייצרים תמורה וחוזר חלילה. את התהליך הזה ניתן לייעל.
בתכנית השלישית נבנה את התצורה, שבה אין מלכה, המאיימת על רעותה, בדרך זו: נבחר מלכה באמצעות היחס select ונבדוק באמצעות היחס attacks3 אם המלכה הזאת מאיימת על אלו שנבחרו קודם וכבר הוצבו במקומות הנכונים. אם המלכה אינה מאיימת על רשימת המלכות שכבר הוצבו, נוסיף אותה לרשימה זו.

להלן התכנית השלישית:

```
queens3(N, Queens) :-
            sequence(N, Sequence),
            solution3(Sequence, [], Queens).
solution3(Sequence, Safe, Queens):-
            select(Queen,Sequence, S1),
            not attacks3(Queen, Safe),
            solution3(S1, [Queen|Safe], Queens).
solution3([], Queens, Queens).
```

sequence - ראה תכנית 10(ב)
select - ראה תכנית 10(ב)
attacks3 - זהה ליחס attacks2 של תכנית 10(ב)

תכנית 10(ג) – פתרון בעית המלכות באמצעות select

תרגיל 10

א. ענה על השאילתות הבאות ועקוב אחר תהליך חישוב התשובה באמצעות ה-trace:

```
?- queens1(Q).
?- queens2(8, Q).
?- queens3(8, Q).
```

(הערה: היעזר בסבלנות, כי חישוב השאילתות עלול להימשך זמן רב).

ב. לפניך שורה של ה-trace, העוקב אחרי חישוב השאילתה שהוצגה על בסיס אחת התכניות שלעיל.

```
[5] call: select(Queen1152,[1,2,3],Queens1153)?
```

ענה על השאלות האלה:
1. מהי השאילתה שאחרי חישובה עוקב ה-trace הזה?
2. על פי איזו מהתכניות בסעיף זה הוצגה השאילתה הזאת?
3. מהי התשובה הצפויה?
נמק את תשובתך.

א'

```
[7] call: solution1([(7 , Y7207),(8 , Y8211)]).
 [8] call: solution1([(8 , Y8211)]).
  [9] call: solution1([]).
  [9] exit: solution1([]).
  [10] call: member(Y1171,[1,2,3,4,5,6,7,8]).
  [10] exit: member(1,[1,2,3,4,5,6,7,8]).
  [10] call: attacks1((8 , 1),[]).
   [11] call: member((A1513 , B1514),[]).
   [11] fail: member((A1513 , B1514),[]).
  [10] fail: attacks1((8 , 1),[]).
 [8] exit: solution1([(8 , 1)]).
 [10] call: member(Y1061,[1,2,3,4,5,6,7,8]).
 [10] exit: member(1,[1,2,3,4,5,6,7,8]).
 [10] call: attacks1((7 , 1),[(8 , 1)]).
  [11] call: member((A1664 , B1665),[(8 , 1)]).
  [11] exit: member((8 , 1),[(8 , 1)]).
  [11] call: 1 = 1.
  [11] exit: 1 = 1.
 [10] exit: attacks1((7 , 1),[(8 , 1)]).
 [10] redo: member(1,[1,2,3,4,5,6,7,8]).
 [10] exit: member(2,[1,2,3,4,5,6,7,8]).
 [10] call: attacks1((7 , 2),[(8 , 1)]).
  [11] call: member((A1668 , B1669),[(8 , 1)]).
  [11] exit: member((8 , 1),[(8 , 1)]).
  [11] call: 1 = 2.
  [11] fail: 1 = 2.
  [11] call: 1 is 2 - 7 + 8.
  [11] fail: 1 is 2 - 7 + 8.
  [11] call: 1 is 2 + 7 - 8.
  [11] exit: 1 is 2 + 7 - 8.
 [10] exit: attacks1((7 , 2),[(8 , 1)]).
 [10] redo: member(2,[1,2,3,4,5,6,7,8]).
 [10] exit: member(3,[1,2,3,4,5,6,7,8]).
 [10] call: attacks1((7 , 3),[(8 , 1)]).

    ≈≈≈≈≈

          [10] fail: attacks1((7 , 3),[(8 , 1)]).
         [7] exit: solution1([(7 , 3),(8 , 1)]).
```

ב'

```
 [10] fail: attacks1((5 , 2),[(6 , 5),(7 , 3),(8 , 1)]).
[5] exit: solution1([(5 , 2),(6 , 5),(7 , 3),(8 , 1)]).
[10] call: member(Y731,[1,2,3,4,5,6,7,8]).
[10] exit: member(1,[1,2,3,4,5,6,7,8]).
[10] call: attacks1((4 , 1),[(5 , 2),(6 , 5),(7 , 3),(8 , 1)]).
 [11] call: member((A2145 , B2146),[(5 , 2),(6 , 5),(7 , 3),(8 , 1)]).
 [11] exit: member((5 , 2),[(5 , 2),(6 , 5),(7 , 3),(8 , 1)]).
 [11] call: 2 = 1.
 [11] fail: 2 = 1.
 [11] call: 2 is 1 - 4 + 5.
 [11] exit: 2 is 1 - 4 + 5.
[10] exit: attacks1((4 , 1),[(5 , 2),(6 , 5),(7 , 3),(8 , 1)]).
[10] redo: member(1,[1;2,3,4,5,6,7,8]).
[10] exit: member(2,[1,2,3,4,5,6,7,8]).
[10] call: attacks1((4 , 2),[(5 , 2),(6 , 5),(7 , 3),(8 , 1)]).
```

איור 21 – חלקים של חישוב השאילתה queens1(Q):
(א) ירידה אל solution1([]) והתחלה של בניית התצורה.
(ב) חלק של תהליך בניית התצורה.

```
tracing: ¦ ?- queens2(5,Queens).
[0] call: queens2(5,Queens81)?
 [1] call: sequence(5,Sequence182).
  [2] call: N1271 is 5 - 1.
  [2] exit: 4 is 5 - 1.
  [2] call: sequence(4,S1274).
   [3] call: N1472 is 4 - 1.
   [3] exit: 3 is 4 - 1.
   [3] call: sequence(3,S1475).
    [4] call: N1673 is 3 - 1.
    [4] exit: 2 is 3 - 1.
    [4] call: sequence(2,S1676).
     [5] call: N1874 is 2 - 1.
     [5] exit: 1 is 2 - 1.
     [5] call: sequence(1,S1877).
     [5] exit: sequence(1,[1]).
     [6] call: append([1],[2],Sequence873).
     [6] exit: append([1],[2],[1,2]).
    [4] exit: sequence(2,[1,2]).
    [6] call: append([1,2],[3],Sequence672).
    [6] exit: append([1,2],[3],[1,2,3]).
   [3] exit: sequence(3,[1,2,3]).
   [6] call: append([1,2,3],[4],Sequence471).
   [6] exit: append([1,2,3],[4],[1,2,3,4]).
  [2] exit: sequence(4,[1,2,3,4]).
  [6] call: append([1,2,3,4],[5],Sequence270).
  [6] exit: append([1,2,3,4],[5],[1,2,3,4,5]).
 [1] exit: sequence(5,[1,2,3,4,5]).
 [6] call: permutation([1,2,3,4,5],Queens181).
  [7] call: select(Y1816,[1,2,3,4,5],NewList1818).
  [7] exit: select(1,[1,2,3,4,5],[2,3,4,5]).
  [8] call: permutation([2,3,4,5],T1817).
   [9] call: select(Y2022,[2,3,4,5],NewList2024).
   [9] exit: select(2,[2,3,4,5],[3,4,5]).
   [10] call: permutation([3,4,5],T2023).
    [11] call: select(Y2228,[3,4,5],NewList2230).
    [11] exit: select(3,[3,4,5],[4,5]).
    [12] call: permutation([4,5],T2229).
     [13] call: select(Y2434,[4,5],NewList2436).
     [13] exit: select(4,[4,5],[5]).
     [14] call: permutation([5],T2435).
      [15] call: select(Y2640,[5],NewList2642).
      [15] exit: select(5,[5],[]).
      [16] call: permutation([],T2641).
       [17] call: select(Y2846,[],NewList2848).
       [17] fail: select(Y2846,[],_2848).
      [16] exit: permutation([],[]).
     [14] exit: permutation([5],[5]).
    [12] exit: permutation([4,5],[4,5]).
   [10] exit: permutation([3,4,5],[3,4,5]).
  [8] exit: permutation([2,3,4,5],[2,3,4,5]).
 [6] exit: permutation([1,2,3,4,5],[1,2,3,4,5]).□
```

≈≈≈≈≈

```
[17] call: solution2([1,2,3,4,5]).
 [18] call: solution2([2,3,4,5]).
  [19] call: solution2([3,4,5]).
   [20] call: solution2([4,5]).
    [21] call: solution2([5]).
     [22] call: solution2([]).
     [22] exit: solution2([]).
     [23] call: attacks2(5,[]).
      [24] call: attacks2(5,[],1).
      [24] fail: attacks2(5,[],1).
     [23] fail: attacks2(5,[]).
    [21] exit: solution2([5]).
    [23] call: attacks2(4,[5]).
     [24] call: attacks2(4,[5],1).
      [25] call: 4 is 5 + 1.
      [25] fail: 4 is 5 + 1.
      [25] call: 4 is 5 - 1.
      [25] exit: 4 is 5 - 1.
     [24] exit: attacks2(4,[5],1).
    [23] exit: attacks2(4,[5]).
    [21] redo: solution2([5]).
     [22] redo: solution2([]).
     [22] fail: solution2([]).
    [21] fail: solution2([5]).
   [20] fail: solution2([4,5]).
  [19] fail: solution2([3,4,5]).
 [18] fail: solution2([2,3,4,5]).
[17] fail: solution2([1,2,3,4,5]).
[6] redo: permutation([1,2,3,4,5],[1,2,3,4,5]).
 [8] redo: permutation([2,3,4,5],[2,3,4,5]).
```

≈≈≈≈≈

```
 [8] exit: permutation([2,3,4,5],[3,5,2,4]).
[6] exit: permutation([1,2,3,4,5],[1,3,5,2,4]).□
[20] call: solution2([1,3,5,2,4]).
```

≈≈≈≈≈

```
      [27] fail: attacks2(1,[3,5,2,4],1).
     [26] fail: attacks2(1,[3,5,2,4]).
    [20] exit: solution2([1,3,5,2,4]).
   [0] exit: queens2(5,[1,3,5,2,4]).
   ... in 26 reductions.

   Queens = [1,3,5,2,4]
```

איור 22 – חישוב השאילתה queens2(5,Q):
התמורה מספר 11 מייצגת את התצורה הנכונה.

```
tracing: ¦ ?- queens3(5,Queens).
[0] call: queens3(5,Queens81)?
 [1] call: sequence(5,Sequence174)?

 ≈≈≈≈≈

 [1] exit: sequence(5,[1,2,3,4,5])?
 [6] call: solution3([1,2,3,4,5],Queens173)?
  [7] call: solution3([1,2,3,4,5],[],Queens1933)?
   [8] call: select(Queen2018,[1,2,3,4,5],S12019)?
   [8] exit: select(1,[1,2,3,4,5],[2,3,4,5])?
   [9] call: attacks2(1,[])?
    [10] call: attacks2(1,[],1)?
    [10] fail: attacks2(1,[],1)?
   [9] fail: attacks2(1,[])?
   [9] call: solution3([2,3,4,5],[1],Queens2017)?
    [10] call: select(Queen2254,[2,3,4,5],S12255)?
    [10] exit: select(2,[2,3,4,5],[3,4,5])?
    [11] call: attacks2(2,[1])?
     [12] call: attacks2(2,[1],1)?
      [13] call: 2 is 1 + 1?
      [13] exit: 2 is 1 + 1.
     [12] exit: attacks2(2,[1],1).
    [11] exit: attacks2(2,[1]).
    [10] redo: select(2,[2,3,4,5],[3,4,5]).
     [11] call: select(X2349,[3,4,5],Z2354).
     [11] exit: select(3,[3,4,5],[4,5]).
    [10] exit: select(3,[2,3,4,5],[2,4,5]).
    [12] call: attacks2(3,[1]).
     [13] call: attacks2(3,[1],1).
      [14] call: 3 is 1 + 1.
      [14] fail: 3 is 1 + 1.
      [14] call: 3 is 1 - 1.
      [14] fail: 3 is 1 - 1.
      [14] call: D12679 is 1 + 1.
      [14] exit: 2 is 1 + 1.
      [14] call: attacks2(3,[],2).
      [14] fail: attacks2(3,[],2).
      [14] redo: 2 is 1 + 1.
      [14] fail: _2679 is 1 + 1.
     [13] fail: attacks2(3,[1],1).
    [12] fail: attacks2(3,[1]).
    [12] call: solution3([2,4,5],[3,1],Queens2253).
     [13] call: select(Queen2598,[2,4,5],S12599).
     [13] exit: select(2,[2,4,5],[4,5]).

     ≈≈≈≈≈

       [20] fail: attacks2(4,[2,5,3,1]).
       [20] call: solution3([],[4,2,5,3,1],Queens3275).
        [21] call: select(Queen3497,[],S13498).
        [21] fail: select(Queen3497,[],_3498).
       [20] exit: solution3([],[4,2,5,3,1],[4,2,5,3,1]).
      [18] exit: solution3([4],[2,5,3,1],[4,2,5,3,1]).
     [16] exit: solution3([2,4],[5,3,1],[4,2,5,3,1]).
    [12] exit: solution3([2,4,5],[3,1],[4,2,5,3,1]).
   [9] exit: solution3([2,3,4,5],[1],[4,2,5,3,1]).
  [7] exit: solution3([1,2,3,4,5],[],[4,2,5,3,1]).
 [6] exit: solution3([1,2,3,4,5],[4,2,5,3,1]).
[0] exit: queens3(5,[4,2,5,3,1]).
... in 21 reductions.

Queens = [4,2,5,3,1]
```

איור 23 – חישוב השאילתה queens3(5,Q):
בניית התצורה הנכונה

```
| ?- queens3(8,Queens).
Queens = [4,2,7,3,6,8,5,1] ;
Queens = [5,2,4,7,3,8,6,1] ;
Queens = [3,5,2,8,6,4,7,1] ;
Queens = [3,6,4,2,8,5,7,1] ;
Queens = [5,7,1,3,8,6,4,2] ;
Queens = [4,6,8,3,1,7,5,2] ;
Queens = [3,6,8,1,4,7,5,2] ;
Queens = [5,3,8,4,7,1,6,2] ;
Queens = [5,7,4,1,3,8,6,2] ;
Queens = [4,1,5,8,6,3,7,2] ;
Queens = [3,6,4,1,8,5,7,2] ;
Queens = [4,7,5,3,1,6,8,2] ;
Queens = [6,4,2,8,5,7,1,3] ;
Queens = [6,4,7,1,8,2,5,3] ;
Queens = [1,7,4,6,8,2,5,3] ;
Queens = [6,8,2,4,1,7,5,3] ;
Queens = [6,2,7,1,4,8,5,3] ;
Queens = [4,7,1,8,5,2,6,3] ;
Queens = [5,8,4,1,7,2,6,3] ;
Queens = [4,8,1,5,7,2,6,3] ;
Queens = [2,7,5,8,1,4,6,3] ;
Queens = [1,7,5,8,2,4,6,3] ;
Queens = [2,5,7,4,1,8,6,3] ;
Queens = [4,2,7,5,1,8,6,3] ;
Queens = [5,7,1,4,2,8,6,3] ;
Queens = [6,4,1,5,8,2,7,3] ;
Queens = [5,1,4,6,8,2,7,3] ;
Queens = [5,2,6,1,7,4,8,3] ;
Queens = [6,3,7,2,8,5,1,4] ;
Queens = [2,7,3,6,8,5,1,4] ;
Queens = [7,3,1,6,8,5,2,4] ;
Queens = [5,1,8,6,3,7,2,4] ;
Queens = [1,5,8,6,3,7,2,4] ;
Queens = [3,6,8,1,5,7,2,4] ;
Queens = [6,3,1,7,5,8,2,4] ;
Queens = [7,5,3,1,6,8,2,4] ;
Queens = [7,3,8,2,5,1,6,4] ;
Queens = [5,3,1,7,2,8,6,4] ;
Queens = [2,5,7,1,3,8,6,4] ;
Queens = [3,6,2,5,8,1,7,4] ;
Queens = [6,1,5,2,8,3,7,4] ;
Queens = [8,3,1,6,2,5,7,4] ;
Queens = [2,8,6,1,3,5,7,4] ;
Queens = [5,7,2,6,3,1,8,4] ;
Queens = [3,6,2,7,5,1,8,4] ;
Queens = [6,2,7,1,3,5,8,4] ;
Queens = [3,7,2,8,6,4,1,5] ;
Queens = [6,3,7,2,4,8,1,5] ;
Queens = [4,2,7,3,6,8,1,5] ;
Queens = [7,1,3,8,6,4,2,5] ;
Queens = [1,6,8,3,7,4,2,5] ;
Queens = [3,8,4,7,1,6,2,5] ;
Queens = [6,3,7,4,1,8,2,5] ;
Queens = [7,4,2,8,6,1,3,5] ;
Queens = [4,6,8,2,7,1,3,5] ;
Queens = [2,6,1,7,4,8,3,5] ;
Queens = [2,4,6,8,3,1,7,5] ;
Queens = [3,6,8,2,4,1,7,5] ;
Queens = [6,3,1,8,4,2,7,5] ;
Queens = [8,4,1,3,6,2,7,5] ;
Queens = [4,8,1,3,6,2,7,5] ;
Queens = [2,6,8,3,1,4,7,5] ;
Queens = [7,2,6,3,1,4,8,5] ;
Queens = [3,6,2,7,1,4,8,5] ;
Queens = [4,7,3,8,2,5,1,6] ;
Queens = [4,8,5,3,1,7,2,6] ;
Queens = [3,5,8,4,1,7,2,6] ;
Queens = [4,2,8,5,7,1,3,6] ;
Queens = [5,7,2,4,8,1,3,6] ;
Queens = [7,4,2,5,8,1,3,6] ;
Queens = [8,2,4,1,7,5,3,6] ;
Queens = [7,2,4,1,8,5,3,6] ;
Queens = [5,1,8,4,2,7,3,6] ;
Queens = [4,1,5,8,2,7,3,6] ;
Queens = [5,2,8,1,4,7,3,6] ;
Queens = [3,7,2,8,5,1,4,6] ;
Queens = [3,1,7,5,8,2,4,6] ;
Queens = [8,2,5,3,1,7,4,6] ;
Queens = [3,5,2,8,1,7,4,6] ;
Queens = [3,5,7,1,4,2,8,6] ;
Queens = [5,2,4,6,8,3,1,7] ;
Queens = [6,3,5,8,1,4,2,7] ;
Queens = [5,8,4,1,3,6,2,7] ;
Queens = [4,2,5,8,6,1,3,7] ;
Queens = [4,6,1,5,2,8,3,7] ;
Queens = [6,3,1,8,5,2,4,7] ;
Queens = [5,3,1,6,8,2,4,7] ;
Queens = [4,2,8,6,1,3,5,7] ;
Queens = [6,3,5,7,1,4,2,8] ;
Queens = [6,4,7,1,3,5,2,8] ;
Queens = [4,7,5,2,6,1,3,8] ;
Queens = [5,7,2,6,3,1,4,8]
```

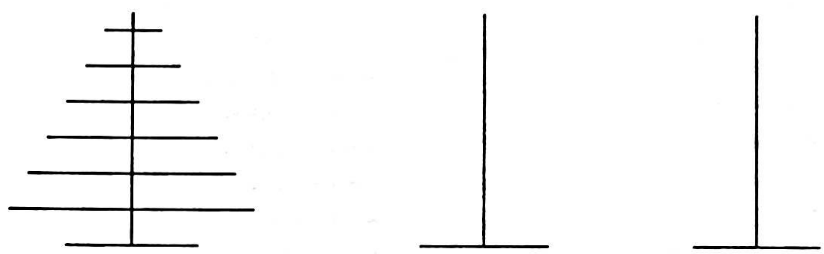
איור 24 – בעיית שמונה המלכות: 92 פתרונות

11. מגדלי האנוי

חומר רקע: פרקים 5, 7, 8, 9, 12 (מתוך פרק 12 נדרשת היכרות עם הפרדיקטים nl, write).

מגדלי האנוי הוא משחק שבו משתמשים בשלושה עמודים וקבוצת דיסקיות, בעלות קטרים שונים המנוקבות במרכזן בדומה לתקליט. העמודים עומדים בשורה אחת והדיסקיות מושחלות על העמוד השמאלי (ראה איור 25). הדיסקיות מסודרות על העמוד, כך שקוטריהן הולכים וגדלים ככל שיורדים כלפי מטה. את הדיסקיות יש להעביר אל העמוד הימני תוך שמירה על המגבלות האלה:

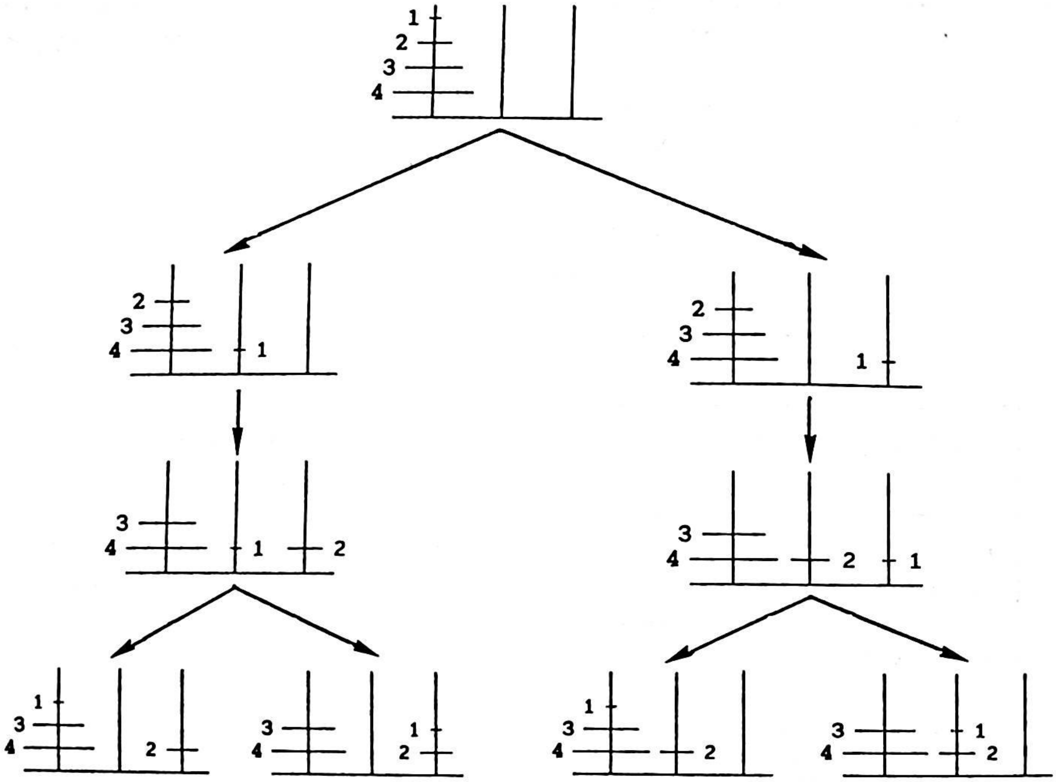
1. מותר להעביר מעמוד לעמוד רק את הדיסקית העליונה.
2. אסור להניח דיסקית כלשהי על גבי דיסקית, הקטנה ממנה בקוטרה.
3. בעת המשחק מונחת כל אחת מהדיסקיות על אחד העמודים. העמוד האמצעי משמש לאחסון הדיסקיות בשלבי הביניים. הוא פנוי מדיסקיות בתחילת המשחק ובסיומו.



איור 25 – מגדלי האנוי

מקורו של המשחק באגדה מן המזרח הרחוק. האגדה מספרת, כי בעת בריאת העולם הוצבו בו שלושה עמודים גבוהים. על העמוד השמאלי הושחלו 64 דיסקיות, אשר סודרו לפי גודל קוטריהן. על הנזירים במנזר נידח הוטלה המשימה להעביר את הדיסקיות אל העמוד הימני בהתאם למגבלות שצויינו. הוסבר לנזירים, כי אם יסטו ממגבלה כלשהי, יהיה עליהם להתחיל את המשימה מחדש. עם השלמת משימתם יתמוטט העולם, ייתמו החיים הארציים, ובני-האדם יזכו בנירוונה ובחיי נצח.

נתחיל את העברת הדיסקיות מהדיסקית העליונה. ניתן להעביר אותה לעמוד האמצעי או לעמוד הימני. איור 26 מדגים את האפשרויות הרבות שלפנינו. את הסדר הנכון שעל פיו יש להעביר את הדיסקיות ניתן לבחור בשיטת ניסוי וטעיה, אבל השיטה הזאת אינה יעילה.



איור 26 – העברת הדיסקיות

נכתוב תכנית, אשר תקבע את הסדר הנכון של העברת הדיסקיות. התכנית תהיה רקורסיבית ותיישם את השיקולים האלה:

א. הפעולה "הקשה ביותר" של המשחק היא העברת הדיסקית האחרונה, הגדולה ביותר. למה? כי כדי לחשוף אותה, יש למלא אחר כל מגבלותיו של המשחק כלפי שאר הדיסקיות. לכן, על העברת הדיסקית האחרונה חל מספר המגבלות הרב ביותר.

ב. מאידך, קיים מצב אחד בלבד, בו ניתן לבצע את העברת הדיסקית האחרונה. זהו המצב, שבו העמוד הימני ריק, שהרי אי אפשר להניח את הדיסקית האחרונה הגדולה ביותר, על דיסקית אחרת כלשהי.

ג. יוצא אם כן, שבעת העברת הדיסקית האחרונה כל הדיסקיות האחרות נמצאות על העמוד האמצעי. בהתאם לכללי המשחק, הן מסודרות על העמוד הזה לפי קוטריהן.

השיקולים האלה מצביעים על אפשרות של תכנית רקורסיבית, שכן העברת מספר כלשהו N של הדיסקיות מושתת על שלב קודם, שבו מועברות N-1 דיסקיות. אין אנו יודעים כיצד נעביר N-1 דיסקיות, אך התכנית הרקורסיבית משיבה על כך באמצעות "נסיגה" אל N-2 דיסקיות, וכן הלאה. בסופו של דבר נגיע לשלב, בו יהיה עלינו להעביר דיסקית אחת בלבד מעמוד לעמוד. שלב זה ישמש "תנאי הגבול" בתכנית.

את התכנית נתחיל בהגדרת היחס hanoi(N), שבו N הוא מספר הדיסקיות המושחלות על העמוד השמאלי:

```
hanoi(N):- replace(N, left, right, middle).
```

היחס replace(N, A, B, C) מתאר העברה של N דיסקיות מעמוד A לעמוד B, תוך שמירה על מגבלות המשחק. העמוד C משמש לאחסון הדיסקיות בשלבי הביניים:

```
replace(N, A, B, C):-
                N1 is N-1,
                replace(N1, A, C, B),
                move(A, B),
                replace(N1, C, B, A).
```

נסביר את המשמעות של כל אחד מהיחסים, הנמצאים בגופו של החוק replace/4:

כדי להעביר N דיסקיות מ-A ל-B תוך שמירת הכללים	replace(N,A,B,C)
יש להעביר N-1 דיסקיות מ-A ל-C, תוך שמירת הכללים	N1 is N-1 replace(N1,A,C,B)
יש להעביר את הדיסקית ה-N-ית האחרונה מ-A ל-B	move(A,B)
יש להעביר N-1 דיסקיות מ-C ל-B, תוך שמירת הכללים	replace(N1,C,B,A)

"תנאי הגבול" הוא העברת דיסקית אחת:

```
replace(1, A, B, _) :- move(A, B).
```

היחס move(A, B) מתאר את העברת הדיסקית האחרונה מעמוד A לעמוד B. על המסך מופיעה הודעה המתארת את העברת הדיסקית:

```
move(A, B):-
         write('move disk from  '), write(A),
         write(' to '), write(B), nl.
```

להלן התכנית בשלמותה:

```
hanoi(N) :-
        replace(N, left, right, middle).
replace(1, A, B, _) :-
        move(A, B).
replace(N, A, B, C) :-
        N1 is N-1,
        replace(N1, A, C, B),
        move(A, B),
        replace(N1, C, B, A).
move(A, B):-
        write('move disk from '), write(A),
        write(' to '), write(B), nl.
```

תכנית 11(א)-מגדלי האנוי

את סדר העברת הדיסקיות ניתן להציג באמצעות רשימה. הנה גירסה נוספת של התכנית, שבה המשתנה Moves מייצג את רשימת התנועות, שעל פיה יש להעביר את הדיסקיות.

```
hanoi(N, Moves) :-
        replace(N, left, right, middle, Moves).
replace(1, A, B, _, [move(A, B)]).
replace(N, A, B, C, Moves) :-
        N1 is N-1,
        replace(N1, A, C, B, X),
        replace(N1, C, B, A, Y),
        append(X, [move(A, B)|Y], Moves).
```

נציג את השאילתה hanoi(4,Moves)-?. בתשובה נקבל את הרשימה Moves. הרשימה תופיעה על המסך ברצף. נערוך אותה באמצעות הפרוצדורה writelist. להלן התכנית בשלמותה:

```
hanoi1(N):-
        hanoi(N, Moves), writelist(Moves).
hanoi(N, Moves) :-
        replace(N, left, right, middle, Moves).
replace(1, A, B, _, [move(A, B)]).
replace(N, A, B, C, Moves) :-
        N1 is N-1,
        replace(N1, A, C, B, X),
        replace(N1, C, B, A, Y),
        append(X, [move(A, B)|Y], Moves).

writelist([]).
writelist([X|T]):-
        write(X),
        nl,
        writelist(T).
```

append - ראה פרק 9

תכנית 11(ב) –
מגדלי האנוי (גירסה שניה)

באיור 27 הוצגו שאילתות על בסיס התכניות 11(א) ו-11(ב). באיור 28 הוצג החישוב של השאילתה hanoi(2). בנספח ג מוצגת גירסה נוספת של "מגדלי האנוי", אשר נכתבה ב-TIProlog.

```
| ?- hanoi(4).
move disk from left to middle
move disk from left to right
move disk from middle to right
move disk from left to middle
move disk from right to left
move disk from right to middle
move disk from left to middle
move disk from left to right
move disk from middle to right
move disk from middle to left
move disk from right to left
move disk from middle to right
move disk from left to middle
move disk from left to right
move disk from middle to right
```

```
| ?- hanoi1(4).
move(left,middle)
move(left,right)
move(middle,right)
move(left,middle)
move(right,left)
move(right,middle)
move(left,middle)
move(left,right)
move(middle,right)
move(middle,left)
move(right,left)
move(middle,right)
move(left,middle)
move(left,right)
move(middle,right)
```

איור 27 – העברת הדיסקיות על-פי תכניות 11(א), 11(ב)

```
tracing: | ?- hanoi(2).
[0] call: hanoi(2)?
 [1] call: replace(2,left,right,middle).
  [2] call: N1259 is 2 - 1.
  [2] exit: 1 is 2 - 1.
  [2] call: replace(1,left,middle,right).
   [3] call: move(left,middle).
    [4] call: write(move disk from ).
move disk from     [4] exit: write(move disk from ).
    [4] call: write(left).
left    [4] exit: write(left).
    [4] call: write( to ).
 to     [4] exit: write( to ).
    [4] call: write(middle).
middle    [4] exit: write(middle).
    [4] call:


    [4] exit:

   [3] exit: move(left,middle).
  [2] exit: replace(1,left,middle,right).
  [4] call: move(left,right).
   [5] call: write(move disk from ).
move disk from    [5] exit: write(move disk from )
   [5] call: write(left).
left   [5] exit: write(left).
   [5] call: write( to ).
 to    [5] exit: write( to ).
   [5] call: write(right).
right   [5] exit: write(right).
   [5] call:

       [7] exit:

      [6] exit: move(middle,right).
     [5] exit: replace(1,middle,right,left).
    [1] exit: replace(2,left,right,middle).
   [0] exit: hanoi(2).
   ... in 7 reductions.
```

איור 28 – חישוב של hanoi(2)

א. תאר באמצעות עץ את כל אחד מהביטויים האלה:

```
1) replace(1, left, right, middle, move[left, right]).
2) append([move(left, right)], [move(left, middle),
                    move(right, middle)], Moves).
3) replace(2, left, middle, right, [move(left, right);
            move(left, middle), move(right, middle)]).
```

ב. מבלי להריץ את התכנית, מצא את התשובה לכל אחת מהשאילתות הבאות:

```
?- hanoi(1).
?- hanoi1(1).
?- hanoi(2).
?- hanoi1(2).
```

ג. מצא את התשובה לשאילתות האלה:

```
?- replace(4, X, Y, Z).
?- replace(4, X, Y, Z, U).
```

12. תוכנה לימודית באמצעות פרולוג – התפתחות של בעלי-חיים

חומר רקע: פרקים 5, 7, 11, 12 (מתוך פרק זה נדרש להכיר את היחסים write ו-nl).

בסעיף זה נדון בתכנית בעלת תוכן לימודי. התכנית מתארת את שלבי ההתפתחות של בעלי חיים שונים ומשלבת ידע בפרולוג עם ידע בתחום ייחודי.

התכנית הינה מודולרית ונכתבת בשיטה "מלמעלה למטה" (top-down). המודול של החוקים מכיל שני יחסים. היחס הראשון life_cycle מגדיר את נושא הדיון Object, ומתאר את השלב הראשון של התפתחותו. היחס השני show_next_state מוגדר באמצעות חוק רקורסיבי. הוא מתאר שרשרת של שלבי ההתפתחות של בעל החיים Object.

את מודול החוקים הבאנו בשתי גירסאות. גירסה א' הינה הצהרתית. בגירסה ב' ביצענו שינוי פרוצדורלי באמצעות Cut.

את מודול העובדות הרכבנו מבלוקים, אשר כל אחד מהם מתאר בעל-חיים מסוים. כתיבת העובדות נושאת אופי לימודי בשל הידע הנדרש בתחום התפתחות של בעלי החיים. את מודול העובדות ניתן להמשיך ולפתח, ככל שיידרש.

בעת דיון בתכנית ניתן להיעזר במילון הבא:

mature	-	מפותח, בוגר, בשל
frog	-	צפרדע
tadpole	-	ראשן (צפרדע בלתי מפותח)
butterfly	-	פרפר
caterpillar	-	זחל, תולעת
cocoon	-	גולם, פקעת
embryo	-	עובר
puppy	-	כלבלב

להלן התכנית בשלמותה. כדי להריצה, צריך להטעין את מודול א' ומודול ג', או לחילופין, את מודול ב' ומודול ג'.

```
/*                    Rules, Version 1            */
life_cycle1(Object) :-
            begins_as(Object,State),
            write(Object),
            write(' : '), write(State),
            show_next_state1(Object,State,NextState).
show_next_state1(Object,State,NextState) :-
            comes_next(Object,State,NextState),
            not mature(NextState),
            write(' ==> '), write(Next),
            show_next_state1(Object,NextState,Next).
show_next_state1(Object,State,NextState) :-
            comes_next(Object,State,NextState),
            mature(NextState),
            write(' ==> '), write(NextState), nl.
```

א. התפתחות של בעלי-חיים, מודול של חוקים, גירסה א'

```
/*                    Rules, Version 2            */
life_cycle2(Object) :-
            begins_as(Object,State),
            write(Object),
            write(': '),
            write(State),
            show_next_state2(Object,State,NextState).

show_next_state2(Object,State,NextState):-
            comes_next(Object,State,NextState),
            write(' ==> '), write(NextState),
            not mature(NextState), !,
            show_next_state2(Object,NextState,Next);
            nl.
```

ב. התפתחות של בעלי-חיים, מודול של חוקים, גירסה ב'

```
/*                         Data                          */
mature(frog).
begins_as(frog,egg).
comes_next(frog,egg,tadpole).
comes_next(frog,tadpole,frog).

mature(butterfly).
begins_as(butterfly,egg).
comes_next(butterfly,egg,caterpillar).
comes_next(butterfly,caterpillar,cocoon).
comes_next(butterfly,cocoon,butterfly).

mature(dog).
begins_as(dog,embryo).
comes_next(dog,embryo,puppy).
comes_next(dog,puppy,dog).

mature(man).
begins_as(man,embryo).
comes_next(man,embryo,newborn).
comes_next(man,newborn,baby).
comes_next(man,baby,boy).
comes_next(man,boy,man).
```

ג. מודול של עובדות

תכנית 12 – התפתחות של בעלי-חיים, מודולים א׳, ב׳, ג׳.

נציג שאילתה ונעיין בתשובה:

```
?- life_cycle1(frog).
frog: egg ==> tadpole ==> frog
```

תשובה זהה נקבל גם כאשר נציג את השאילתה life_cycle2(frog).

את התכנית הזאת ניתן להסב לתחומי לימוד שונים. ניתן לתאר באמצעותה את התפתחות החברה, התפתחות רוחנית של האדם, התפתחויות פוליטיות שונות, שלבי פיתוח חומרים בתהליכים כימיים או פיסיקליים, ועוד.

את התכנית "התפתחות של בעלי-חיים" פיתחה גילה רז, סטודנטית בקורס לפרולוג.

תרגיל 12

א. בהסתמך על המודולים א׳ ו-ג׳ של התכנית, תאר באמצעות הפרולוג את שלבי ההתפתחות של שילטון במדינות שונות. את היחס mature החלף ביחס current (נוכחי).

ב. נציג שאילתות בנושא התפתחות בעלי-חיים ונעיין בתשובות (ראה איור 29). תחילה נציג את השאילתה life_cycle1(X). בתשובה נקבל את השרשרת ואת ההצבה תחת X. לעומת זאת, כאשר נציג את השאילתה life_cycle2(X). ההצבה תחת X שמופיעה אחרי השרשרת תחזור על עצמה מספר פעמים.
הסבר את התופעה.

```
| ?- life_cycle1(X).
frog : egg ==> tadpole ==> frog

X = frog ;
butterfly : egg ==> caterpillar ==> cocoon ==> butterfly

X = butterfly ;
dog : embrio ==> puppy ==> dog

X = dog ;
man : embrio ==> newborn ==> baby ==> boy ==> man

X = man


| ?- life_cycle2(X).
frog: egg ==> tadpole ==> frog

X = frog ;

X = frog ;
butterfly: egg ==> caterpillar ==> cocoon ==> butterfly

X = butterfly ;

X = butterfly ;

X = butterfly ;
dog: embrio ==> puppy ==> dog

X = dog ;

X = dog ;
man: embrio ==> newborn ==> baby ==> boy ==> man

X = man ;

X = man ;

X = man ;

X = man
```

איור 29 – תשובות לשאילתות life_cycle1(X)

נציין, כי מערכת הפרולוג מארגנת את מודול העובדות על-פי פרוצדורות, ולא על-פי בלוקים (ראה איור 30).

```
| ?- listing.

show_next_state1(Object,State,NextState) :-
   comes_next(Object,State,NextState),
    not mature(NextState),
   write(' ==> '),
   write(NextState),
   show_next_state1(Object,NextState,Next).
show_next_state1(Object,State,NextState) :-
   comes_next(Object,State,NextState),
   mature(NextState),
   write(' ==> '),
   write(NextState),
   nl.

show_next_state2(Object,State,NextState):-
                                 comes_next(Object,State,NextState),
                                 write(' ==> '),
                                 write(NextState),
                                 not mature(NextState),
                                 !,
                                 show_next_state2(Object,NextState,Next);
                                 nl.

begins_as(frog,egg).
begins_as(butterfly,egg).
begins_as(dog,embrio).
begins_as(man,embrio).
mature(frog).
mature(butterfly).
mature(dog).
mature(man).
comes_next(frog,egg,tadpole).
comes_next(frog,tadpole,frog).
comes_next(butterfly,egg,caterpillar).
comes_next(butterfly,caterpillar,cocoon).
comes_next(butterfly,cocoon,butterfly).
comes_next(dog,embrio,puppy).
comes_next(dog,puppy,dog).
comes_next(man,embrio,newborn).
comes_next(man,newborn,baby).
comes_next(man,baby,boy).
comes_next(man,boy,man).
life_cycle1(Object) :-
   begins_as(Object,State),
   write(Object),
   write(' : '),
   write(State),
   show_next_state1(Object,State,NextState).
life_cycle2(Object) :-
   begins_as(Object,State),
   write(Object),
   write(': '),
   write(State),
   show_next_state2(Object,State,NextState).
```

איור 30 – ארגון של הנתונים על-ידי המערכת של פרולוג

13. מערכת מומחה – גן החיות

חומר רקע: פרקים 5, 7, 8, 11, 12.

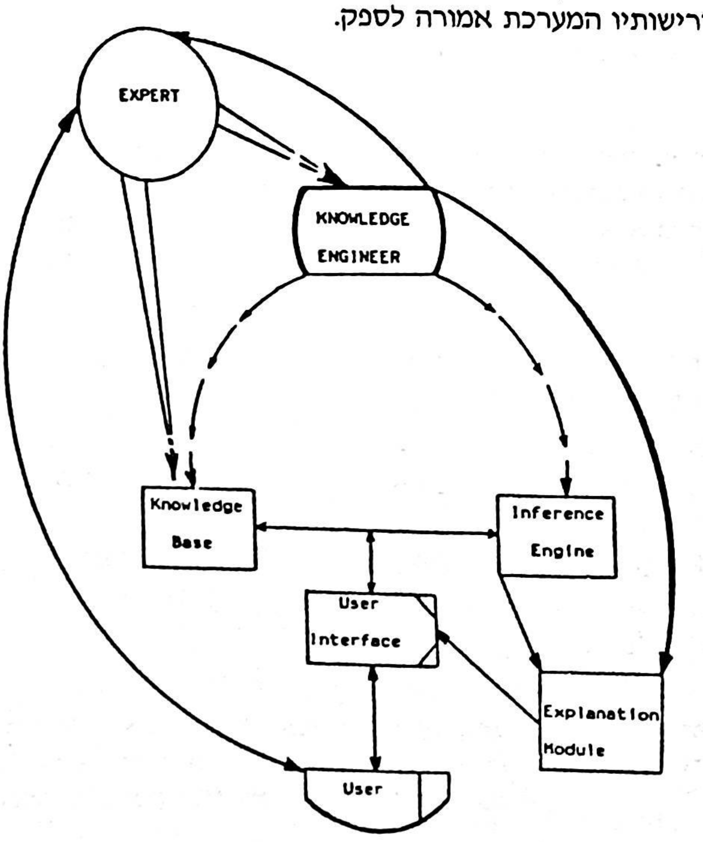
בסעיף זה נציג מערכת מומחה לימודית בשם "גן החיות". מערכת מומחה הינה תכנית מחשב לפתרון בעיות בתחום מסוים, אשר:

* כוללת ידע רב של מומחה בתחום זה,
* נעזרת באסטרטגיות של המומחה כדי לפתור בעיות בתחום זה,
* הינה הידברותית (אינטראקטיבית), משמע מתחקרת את המשתמש ומסבירה את שיקוליה לפי בקשתו.

מרכיב חשוב של מערכת מומחה הינו **מסד ידע** (Knowledge Base), המכיל ידע של מומחה בתחום מומחיותו. מרכיב חשוב נוסף הינו **מנגנון הסקה** (Inference Engine), אשר באמצעותו המערכת מגיעה למסקנות תוך שימוש במסד ידע ועל פי תיחקור של המשתמש. המערכת מבצעת את ההידברות עם המשתמש באמצעות **מישק המשתמש** (User's Interface). אלה הם המרכיבים הבסיסיים של מערכת מומחה. במערכות מומחה ספציפיות ישנם מרכיבים נוספים, בהתאם למבנה המערכת ובהתאם לדרישות המשתמש.

מערכת מומחה נבנית על ידי צוות. נציין שלושה מחברי הצוות:

* המומחה, שאת מומחיותו המערכת אמורה להנציח.
* מהנדס ידע, אשר מעצב באופן פורמלי נאות את הידע של המומחה.
* המשתמש, שאת דרישותיו המערכת אמורה לספק.



איור 31 – מבנה של מערכת מומחה

נבנה עתה מערכת מומחה לזיהוי חיות על פי סימנים שונים. המומחה שלנו הוא רובוט נחמד בשם רובי, הנמצא בספר Artificial Intelligence של Winston. בתור מהנדסי ידע נלהבים, נעסוק בנצחת ידע של רובי באמצעות מערכת מומחה. תחילה נקשיב למונולוג (חד-שיח) של רובי, ומכיון שהוא דובר אנגלית, ניעזר במילון הבא:

carnivore	-	חיה האוכלת בשר
mammal	-	יונק
pointed teeth	-	שיניים בולטות
claw	-	טפר, ציפורן
tawny color	-	צבע צהוב-חום
dark spots	-	כתמים כהים

להלן המונולוג של רובי:

A Protocol for Robbie

Let's see, I want to identify this animal, I wonder if it's a cheetah – I have nothing to go on yet, so I think I'll start by looking into the possibility that it is a carnivore – I had better check to see if it is a mammal first – yes, that is ok hair – it doesn't seem to be eating anything so I can't tell if it is a carnivore, – but oh, yes, it has pointed teeth, claws, and forward-pointing eyes, so it is a carnivore all right – now where was I – it's a carnivore, and I also see that it has a tawny color and dark spots – surely it must be a cheetah!

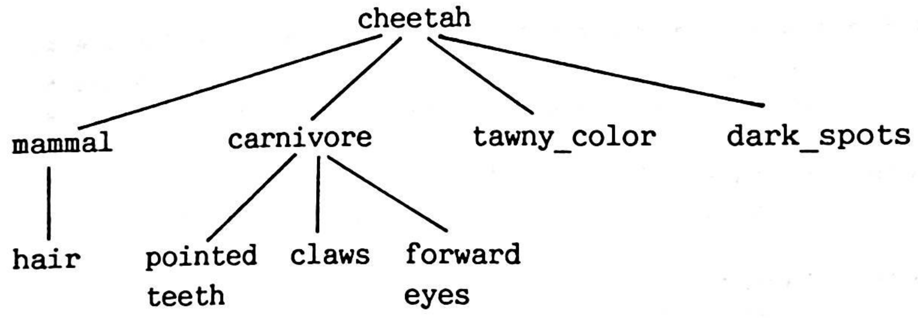
P. Winston, Artificial Intelligence, p. 202,
Addison-Wesley, 1984

כדי להבין את האסטרטגיה שעל פיה רובי מזהה את החיה, נעיין בשרשרת שיקוליו. אליבא דרובי, החיה היא ציטה אם היא יונק, וגם אוכלת בשר, וגם יש לה סימנים מיוחדים כגון צבע צהוב-חום וכתמים כהים. היא יונק אם יש לה שערות. היא אוכלת בשר, אם יש לה ציפורניים חדות ועיניים ושיניים בולטות. הנה שרשרת שיקוליו של רובי:

has it hair?	yes	(==> mammal)
has it pointed-teeth?	yes	
has it claws?	yes	
has it forward-eyes?	yes	(==> carnivore)
has it tawny-color?	yes	
has it dark-spots?	yes	(==> cheetah)

איור 32 – שרשרת שיקוליו של רובי

רובי מארגן את שיקוליו באמצעות שיטה, המכונה בשם **שרשרת הוכחה לאחור** (backward chaining). הוא מציג לעצמו את המטרה: ציטה, ולאחר מכן בודק אם החיה שמולו תואמת לתאור של ציטה. נציג את מערכת שיקוליו בצורת עץ:



איור 33 – הצגת שיקוליו של רובי באמצעות עץ

אחרי כמה שיחות התברר לנו, כי את הכללים של רובי ניתן להציג באמצעות הביטויים שצורתם "אם...אז...". לכן סוכם, כי שפת הפרולוג היא המתאימה ביותר כדי להציג את הידע של רובי. על כן כתבנו בפרולוג את מסד הידע ואת מישק המשתמש. המערכת של פרולוג סיפקה את מנגנון ההסקה (אשר הוא כידוע, אחד מנכסיה הבנויים). בדרך זו השלמנו בנייה של מערכת מומחה לזיהוי חיות. להלן המערכת:

תכנית 13 -

א. מישק המשתמש: כניסה למערכת, תישאול המשתמש, ניהול דינמי של הנתונים.

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                              */
/*                  מערכת מומחה - גן החיות                      */
/*                                                              */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* כניסה למערכת: הזמנת המומחה נעשית באמצעות השאילתה zoo -? */

   zoo:-
        write('WELCOME TO ZOO'), nl,
        write('My name is Robbie and'),nl,
        write('I shall help you to identify this animal'),
        nl,nl,write('When answering my question'),nl,
        write('please type yes or no'),nl,nl,
        animal_is(X),!,nl,
        write('I think,your animal is '),write(X),
        nl,write('If you want to continue'),
        write(' please query:  ?-clear—facts.'),nl,
        write(' AND  GO  AHEAD  WITH  zoo '),nl,nl.

   zoo:-
        write('I am confused!  I do not know'),
        write( ' who your animal is.'),nl,
        write('DO YOU WANT TO ASK ME AGAIN? '),clear_facts.
```

```
/* * * * * * * *  * * * * * * * * * * * * * * * * * * * * */
/*           clear_facts ניהול דינמי של הנתונים: באמצעות היחס */
/*         נמחקות תשובות של המשתמש והמערכת מוכנה לדו-שיח חדש */

    clear_facts:-
            abolish(yes_answer,2),fail.

    clear_facts:-
            abolish(no_answer,2),fail.

    clear_facts:-
            write('Please type the dot to enter the top-level'),
            read(_).


/* * * * * * * * * * * * * * * * * * * * * * * * ** * * * * */
/*           תישאול המשתמש: לאחר עיון במסד הידע המערכת מחליטה */
/*                                  מה, ומתי לשאול          */

    positive(X,Y) :- yes_answer(X,Y),!.
    positive(X,Y) :- not(no_answer(X,Y)),ask(X,Y).
    negative(X,Y) :- no_answer(X,Y),!.
    negative(X,Y) :- not(yes_answer(X,Y)),ask(X,Y).

    ask(X,Y):-
       write(X), write(' it '), write(Y), write('?'),
       nl,
       read(Answer),
       respond(X, Y, Answer).


/* * * * * * * * * * * * * * * * * * * * * * * * * * * * **/
/*          ניהול דינמי של הנתונים: את תשובות המשתמש המערכת */
/*                   מוסיפה למסד הידע                      */

     respond(X, Y, yes):-
       asserta(yes_answer(X, Y)).

     respond(X, Y, no):-
       asserta(no_answer(X, Y)),
       fail.
```

ב. מסד ידע: שרשרת שיקוליו של רובי מוצגת באמצעות חוקים של פרולוג.

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * **/
/*          הידע של רובי מוצג באמצעות חוקים של פרולוג          */

animal_is(cheetah):-
   it_is(mammal) ,
   it_is(carnivore),
   positive(has,tawny_color),
   positive(has,dark_spots).

animal_is(tiger):-
   it_is(mammal),
   it_is(carnivore),
   positive(has,tawny_color),
   positive(has,black_stripes).

animal_is(giraffe):-
   it_is(ungulate),
   positive(has,long_neck),
   positive(has,long_legs),
   positive(has,dark_spots).

animal_is(zebra):-
   it_is(ungulate),
   positive(has,black_stripes).

animal_is(ostrich):-
   it_is(bird),
   negative(does,fly),
   positive(has,long_neck),
   positive(has,long_legs),
   positive(has,black_and_white_color).

animal_is(penguin):-
   it_is(bird),
   negative(does,fly),
   positive(does,swim),
   positive(has,black_and_white_color).

animal_is(albatross):-
   it_is(bird),
   positive(does,fly_well).
```

```
it_is(mammal):-
   positive(has,hair).
it_is(mammal):-
   positive(does,give_milk).

it_is(bird):-
   positive(has,feathers).
it_is(bird):-
   positive(does,fly),
   positive(does,lay_eggs).

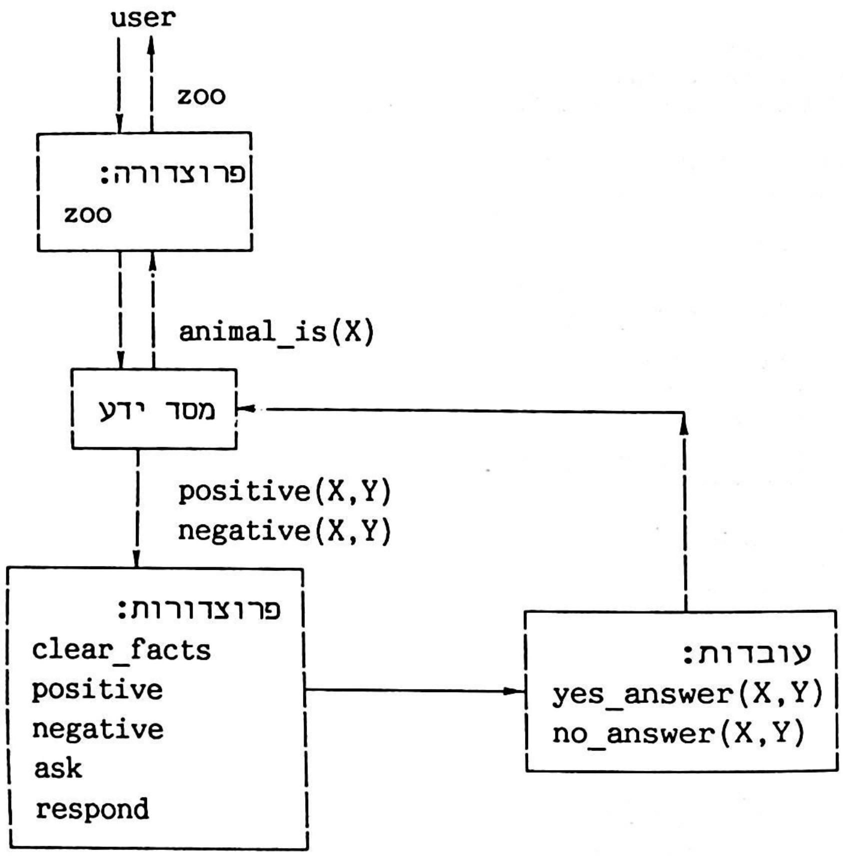
it_is(carnivore):-
   positive(does,eat_meat).
it_is(carnivore):-
   positive(has,pointed_teeth),
   positive(has,claws),
   positive(has,forward_eyes).

it_is(ungulate):-
   it_is(mammal),
   positive(has,hooves).

it_is(ungulate):-
   it_is(mammal),
   positive(does,chew_cud).
```

תכנית 13 – מערכת מומחה לזיהוי חיות:
א. מישק המשתמש
ב. מסד ידע

את המערכת חילקנו לשני מודולים. הפרוצדורה zoo של המודול הראשון מזמינה את המטרה animal_is(X). מתוך המודול השני, המכיל את הידע של רובי, מתבצעת ההצבה cheetah תחת המשתנה X. עתה המערכת בודקת, אם החיה, אשר אותה רואה המשתמש, היא אכן צ׳יטה. כאשר המערכת מגיעה לאחת המטרות positive(X,Y) או negative(X,Y), היא פונה למשתמש כדי להשלים בעזרתו את המידע החסר. לאחר קבלת המידע, המערכת חוזרת למסד הידע. את הנאמר לעיל תיארנו באמצעות איור 34:



איור 34 – הקשרים בין המודולים של המערכת

המערכת פועלת באופן הבא: היא מגדירה את המטרה animal_is(cheetah) ובודקת את הנתונים על מנת לאמת או להכחיש את המטרה הזאת. שיטה זו מכונה בשם backward-chaining או goal-driven. שפת הפרולוג תומכת בשיטה זו.
הבה נשוחח עם רובי בנושא זיהוי החיות. את תוצאות השיחה ריכזנו באיור 35.
נציין, כי רובי הוא מומחה אמיץ, שכן הוא מכיר את מגבלות הידע שלו.

תרגיל 13

א. הצג את השאילתה ?-zoo. ענה בחיוב רק על שאלותיה של המערכת, ששייכות לשרשרת שיקוליו של רובי (איורים 32 ו-33). מה תהיה החלטת המערכת בעניין הזיהוי?

ב. ציין את העובדות ואת החוקים של מערכת מומחה, אשר על פיהם התקבלה התשובה הנ"ל.

ג. בדוק את עקביות השאלות שהמערכת מפנה אל המשתמש. האם יש מקום לשיפורים? נמק ושפר.

```
yes
¦ ?- zoo.
WELCOME TO ZOO
My name is Robbie and
I shall help you to identify this animal

When answering my question
please type yes or no

has it  hair?
¦ no.
does it  give_milk?
¦ yes.
does it  eat_meat?
¦ no.
has it  pointed_teeth?
¦ no.
has it  hooves?
¦ yes.
has it  long_neck?
¦ yes.
has it  long_legs?
¦ yes.
has it  dark_spots?
¦ yes.

I think,your animal is giraffe
If you want to continue please query:  ?-clear_facts.
 AND  GO  AHEAD  WITH  zoo


yes
¦ ?- clear_facts.
Please type the dot to enter the top-level.

yes
¦ ?- zoo.
WELCOME TO ZOO
My name is Robbie and
I shall help you to identify this animal

When answering my question
please type yes or no

has it  hair?
¦ no.
does it  give_milk?
¦ no.
has it  feathers?
¦ no.
does it  fly?
¦ no.
I am confused!  I do not know who your animal is.
DO YOU WANT TO ASK ME AGAIN? Please type the dot to enter the top-level.
```

איור 35 – שיחה עם רובי

14. קונכיה (Shell) של מערכת מומחה

חומר רקע: פרקים 5, 7, 8, 9, 10, 11, 12.

בעת הפעלת מערכת מומחה של סעיף 13 ראינו, כי הפרוצדורות של המודול הראשון, להוציא את הפרוצדורה zoo, כלל לא היו תלויות במודול השני. המודול השני הכיל ידע ספציפי בנושא זיהוי החיות, והמודול הראשון ניהל את הידע הזה מבלי להיות תלוי בו. משמעות הדבר היא, כי המודול הזה מסוגל לנהל כל מסד ידע, בעל מבנה נתונים התואם את המבנה של המודול, ועל כן ניתן להיעזר בו לבניית מערכת מומחה נוספת.
את הנאמר לעיל ניתן גם לתאר בדרך אחרת: ניקח מערכת מומחה, נוציא ממנה את מסד הידע ונשאיר את המודולים ה"מטפלים" בתהליך ההסקה ובהידברות עם המשתמש. מודולים אלו מכונים בשם **קונכיה של מערכת מומחה** (expert system shell). אם נצרף לקונכיה מסד ידע בתחום ספציפי, אשר מבנה הנתונים שלו הותאם למבנה הקונכיה, נקבל מערכת מומחה בתחום זה.

עתה נציג לפניכם קונכיית מערכת מומחה, העונה על השאלות why (מדוע) ו-how (איך) של המשתמש. לשאלות אלו המשתמש נזקק, כאשר הוא רוצה לקבל נימוק והסבר לשיקולי המערכת.

לנימוקים ולהסברים של המערכת ישנם שני צדדים. הצד האחד הינו הצד הסמנטי. מבחינה סמנטית, המערכת "פורשת" בפני המשתמש את תהליך ההסקה הלוגית האופייני לה. לשם כך היא נעזרת בביטויים, כגון: "אני יודע ש-...", או "אתה אמרת לי ש-...", או "אם...אז...", או "כדי להסיק כי..., עלי לדעת את...", וכן הלאה. זהו חלק של תהליך הידברות בין המשתמש לבין המערכת. על המשתמש, השייך לצוות כותבי המערכת (מומחה, מהנדס-ידע, משתמש), חלה חובה להעריך את איכות ההסברים: האם הם סבירים, ברורים, מלאים וכדומה.

הצד האחר של הסברי המערכת הינו הצד הטכני. ברמה טכנית המערכת מציגה חוק מסוים או אוסף של חוקים (תלוי במבנה הקונכיה). חוק זה נדרש על מנת להמשיך בביצוע של תהליך החישוב וכדי להציגו, יש לספק גישה למנגנון החישובי.

הקונכיה, שאותה נציג, נכתבה ב-WISDOM-PROLOG. המפרש (interpreter) של הגירסה הזאת נכתב בפרולוג גם הוא. המפרש הינו תכנית מחשב מיוחדת, אשר תפקידה לתרגם לשפת המכונה תכנית אחרת שנכתבה בשפה עילית. התרגום והביצוע משולבים על ידי המפרש זה בזה, שכן הוא מתרגם הוראה ומיד מבצע אותה. המפרש של שפה עילית, אשר נכתב בשפה עצמה, מכונה בשם **מפרש-על** (metainterpreter). לפיכך, המפרש של WISDOM-PROLOG הינו מפרש-על. באמצעותו מבצעים תכנות-על: העובדות, החוקים והשאילתות של תכנית בפרולוג הינם ארגומנטים (ניתן גם לומר: קלט) של מפרש-על. המפרש-על של פרולוג מיישם את המודל החישובי של תכנית לוגית באמצעות החיפוש לעומק והוא מהווה חלק של הקונכיה שאותה נציג: הוא "אחראי" על תהליך ההסקה. בקונכיה שלהלן נראה שתי גירסות של מפרש-על. ההבדל ביניהן הינו בהצגת החוקים של התכנית המתורגמת, להלן הקונכיה.

```
/* תכנית 14 - * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                          */
/*                  קונכיה של מערכת מומחה                    */
/*                                                          */
/************************************************************/
/* מפרש-על של פרולוג: הפרוצדורה solve פונה לרזולבנטה של Body */
/*       המטרה Goal ומבצעת את החיפוש לעומק                   */
/*                  באמצעות הרקורסיה                         */

solve(Goal) :- solve(Goal,[]).
solve(true,Rules).
solve((Goal1,Goal2),Rules) :-
        solve(Goal1,Rules),solve(Goal2,Rules).
solve(Goal,Rules) :-
        clause(Goal,Body),
        solve(Body,[rule(Goal,Body)|Rules]).
solve(Goal,Rules) :-
        data_needed(Goal),
        not known(Goal),
        ask(Goal,Answer),
        respond(Goal,Answer,Rules).
```

א. **מנגנון ההסקה:** מפרש-על של פרולוג

```
/************************************************************/
/* הצגת שאלות: הפיסקה האחרונה של solve מפנה את התכנית למודול */
/*     ב', המבצע את תישאול המשתמש וניהול הנתונים             */

ask(Goal,Answer) :-
            write(Goal),
            write('?'),
            read(Answer).
respond(Goal,yes,Rules) :-
            assert(Goal).
respond(Goal,no,Rules) :-
            assert(no_answer(Goal)),
            fail.
respond(Goal,why,[Rule|Rest]) :-
            display_rule(Rule),
            ask(Goal,Answer),
            respond(Goal,Answer,Rest).
respond(Goal,why,[]) :-
            write('No more explanation'),
            nl,
            ask(Goal ,Answer),
            respond(Goal,Answer,Rules).

known(Goal) :- Goal.
known(Goal) :- no_answer(Goal).
```

ב. **מישק המשתמש:** הצגת שאלות וניהול דינמי של נתונים.

```
/**********************************************************/
/*        הצגת הסבר: נזכירכם, כי conjunction הינו הכינוי */
/*                        של הקשר AND                    */

display_rule(rule(X,Y)) :-
                 write('IF '),
                 write_conjunction(Y),
                 write('THEN '),
                 write(X),
                 nl
write_conjunction((X,Y)) :-
                 !,
                 write(X),
                 write(' AND '),
                 write_conjunction(Y).
write_conjunction(X) :-
                 write(X),
                 nl.
```

ג. **מישק המשתמש:** הצגת הסבר.

```
/**********************************************************/
/*        המפרש-על: מבנה נתונים שונה מזה שבמודול א'      */

solve1(true,true).
solve1((Goal1,Goal2),(Proof1,Proof2)) :-
           solve1(Goal1,Proof1),
           solve1(Goal2,Proof2).
solve1(Goal,(Goal:-Proof)) :-
           clause(Goal,Body),
           solve1(Body,Proof).
```

ד. **מנגנון ההסקה:** מפרש-על של פרולוג, גירסה נוספת.

```
/*********************************************************/
/* דוגמה נוספת של תכנות-על: הקלט של הפרוצדורות שלהלן הינו */
/*   Proof המטרה Goal, הרזולבנטה                          */
/*                     Fact והעובדה                        */

interpret((Proof1,Proof2)) :-
            interpret(Proof1),
            interpret(Proof2).

interpret(Proof) :-
            fact(Proof,Fact),
            nl,
            write('I KNOW THAT '),
            write(Fact),
            write(' IS A FACT IN MY DATABASE'),
            nl.

interpret(Proof) :-
           rule(Proof,Head,Body,Proof1),
           nl,
           write(Head),
           write(' IS PROVED USING THE RULE '),
           nl,
           display_rule(rule(Head,Body)),
           interpret(Proof1).

fact((Fact:-true),Fact).
rule((Goal:-Proof),Goal,Body,Proof):-
           Proof\=true,
           extract_body(Proof,Body).

extract_body((Proof1,Proof2),(Body1,Body2)) :-
            !,
            extract_body(Proof1,Body1),
            extract_body(Proof2,Body2).
extract_body((Goal:-Proof),Goal).
```

ה. **הצגה של תהליך ההסקה:** דוגמה נוספת של תכנות-על.

```
/*********************************************************/
/* ההוראה "הוכח והצג": השאילתה how(Goal) ?- מציגה למשתמש את */
/*                          ההוכחה                          */

     how(Goal) :-
          solve1(Goal,Proof),
          interpret(Proof).
```

ו. **הזמנת ההצגה:** ההוראה "הוכח והצג".

תכנית 14 – קונכיה של מערכת מומחה, מודולים א׳-ו׳

חומר נוסף בנושא בניית מערכת מומחה על בסיס מפרש-על של פרולוג, ניתן למצוא ב-[Sterling 1983], ב-[Sterling & Shapiro, 1986] וב-[Rossi, 1986].

במאמר של [Golumbic & others, 1986] מתוארת מערכת מומחה, שנבנתה בפרולוג. מערכת זו מייעצת לסטודנטים בהכנת תכנית הלימודים.

```
/**************/

a :- b,c.

b :- d,e.

data_needed(c).
data_needed(d).
data_needed(e).

/*************/
```

```
/******************/
/*                */
/*       a        */
/*      /\        */
/*     /  \       */
/*    b    c      */
/*   / \          */
/*  /   \         */
/* d     e        */
/*                */
/******************/
```

```
| ?- solve(a).
d?why.
IF d AND e
THEN b
d?yes.
e?yes.
c?why.
IF b AND c
THEN a
c?yes.

yes
| ?- how(a).

a IS PROVED USING THE RULE
IF b AND c
THEN a

b IS PROVED USING THE RULE
IF d AND e
THEN b

I KNOW THAT d IS A FACT IN MY DATABASE

I KNOW THAT e IS A FACT IN MY DATABASE

I KNOW THAT c IS A FACT IN MY DATABASE
```

איור 36 – הדגמה של פעולת הקונכיה

```
        /*******          YOUR ADVISER & CO.       ******/
        /*   address:     HOTEL SING-SING ,U.S.A.     */

adviser :-
        solve(recommended_oil_investment),
        write('Oil investment is strongly recommended'),
        nl,write('Maybe you loose, maybe not'),
        nl,write('Happy to serve you'),
        nl,write('Have a nice day'),nl,nl.
adviser :-
        solve(recommended_solid_investment),
        write('Today is October,1,1983'),
        nl,write('I think you shoud buy MNAIOT-HABANKIM'),
        nl,write('Good Luck, and happy to serve you'),nl,nl.
adviser :-
        solve(recommended_vaduz_investment),
        write('Take a vacation, fly to VADUZ'),
        nl,write('A nice place for a tourist'),
        nl,write('And please do not come back'),nl,nl.
adviser :-
        solve(recommended_balata_investment),
        nl,write('HOME, SWEET HOME'),
        nl,write('SWEET BALATA'),
        nl,write('And please do not forget us'),nl,nl.


recommended_oil_investment :-
                          legal_money,
                          risk_tolerance_high.
recommended_solid_investment :-
                          legal_money,
                          risk_tolerance_low.
recommended_vaduz_investment :-
                          illegal_money,
                          risk_tolerance_high.
recommended_balata_investment :-
                          illegal_money,
                          risk_tolerance_low.

risk_tolerance_high :-
                   liquidity_more_than_million_shekel.
risk_tolerance_low :-
                   no_answer(liquidity_more_than_million_shekel).

illegal_money :-
              no_answer(legal_money).

data_needed(legal_money).
data_needed(liquidity_more_than_million_shekel).
```

איור 37 – דוגמה של מסד ידע, התואם את הקונכיה

```
/***    YOUR COSMETICIAN    ***/

cosmetics(Subject, strong) :-
            definition(Subject, lady),
            occasion(Subject,party).
cosmetics(Subject, medium) :-
            definition(Subject, lady),
            occasion(Subject, shopping).
cosmetics(Subject, medium) :-
            definition(Subject, young_lady),
            occasion(Subject, party).
cosmetics(Subject, not_at_all) :-
            definition(Subject, young_lady),
            occasion(Subject, tennis).

data_needed(definition(Subject, lady)).
data_needed(definition(Subject, young_lady)).
data_needed(occasion(Subject, party)).
data_needed(occasion(Subject, shopping)).
data_needed(occasion(Subject, tennis)).
```

```
Wisdom(tm) Prolog interpreter version 2.10
Copyright (C) M. Safra & M.Goldberg(MAXBYTE), Israel
Copyright (C) MIT Press (U.S.A)

yes
¦ ?- [+'a:why', +'a:how', +'a:cosmet.'].
a:why   consulted
a:how   consulted
a:cosmet.   consulted

yes
¦ ?- solve(cosmetics(naomi,X)).
definition(naomi,lady)?no.
definition(naomi,young_lady)?why.
IF definition(naomi,young_lady) AND occasion(naomi,party)
THEN cosmetics(naomi,medium)
definition(naomi,young_lady)?yes.
occasion(naomi,party)?no.
occasion(naomi,tennis)?yes.

X = not_at_all

yes
¦ ?- how(cosmetics(naomi,not_at_all)).

cosmetics(naomi,not_at_all) IS PROVED USING THE RULE
IF definition(naomi,young_lady) AND occasion(naomi,tennis)
THEN cosmetics(naomi,not_at_all)

I KNOW THAT definition(naomi,young_lady) IS A FACT IN MY DATABASE

I KNOW THAT occasion(naomi,tennis) IS A FACT IN MY DATABASE
```

איור 38 – מערכת מומחה Your Cosmetician:

הקובץ cosmet מכיל את מסד הידע.
הקובץ why מכיל את המודולים א', ב', ג' של הקונכיה.
הקובץ how מכיל את המודולים ד', ה', ו'.

תרגיל 14

א. לפניך מסד ידע, המכונה "רפואה שלימה".

```
drug_to_take:-
            solve(acamol),
            write('You should take acamol'),
            nl.

drug_to_take:-
            solve(nussidex),
            write('You should take nussidex'),
            nl.

drug_to_take:-
            write('I have nothing for you'),
            nl.

acamol:- headache,fever.
nussidex:- headache,runny_nose.

data_needed(headache).
data_needed(fever).
data_needed(runny_nose).
```

הפעל את התכנית 14 והצג את השאילתה:

```
?- drug_to_take.
```

אם המערכת המליצה על נטילת אקמול, הצג שאילתה נוספת:

```
?- how(acamol).
```

עקוב באמצעות ה-trace אחרי חישוב השאילתות.

(את מסד הידע drug_to_take כתב נועם נוימן, סטודנט בקורס לפרולוג)

כתוב מסד ידע בתחום מומחיותך.

15. תכניות יישום – תרגול

בסעיף זה נציג אוסף שאלות לתרגול עצמי. לא נציין מהו חומר הרקע עבור כל שאלה ושאלה. אנו מצפים שהקורא יקבע זאת בעצמו כחלק של התרגול.

עבודה מהנה !

שאלה 1

מפתח התכנית "סיפור מן החיים בשפת פרולוג" החליט לשנות את מבנה העובדות:

```
subject(sara,[lady,beautiful,blond_hair,intelligent]).
subject(ester,[lady,tall,blond_hair]).
subject(ruth,[lady,beautiful]).
subject(alice,[lady,intelligent]).
subject(victoria,[lady,gossip,blond_hair]).
subject(patricia,[lady,gossip,tall]).
subject(liz,[lady,gossip,beautiful,tall,blond_hair]).
subject(nataly,[lady,beautiful,intelligent]).
subject(mary,[gossip,tall,intelligent]).
```

א. הכללים של אדון צ׳רלס לא השתנו. שנה את חוקי הפרולוג, כדי לבטא את הכללים של צ׳רלס בהתאם למבנה הנתונים החדש.

ב. כתוב חוק של פרולוג, המגדיר זוגות נשים, אשר תכונותיהן הינן קבוצות זרות (כלומר, נשים שונות לחלוטין).

ג. כתוב חוק של פרולוג, המתאר באמצעות הרשימה List את הנשים של התכנית, אשר אינן אינטליגנטיות.

שאלה 2

עיין בתכנית "חד-גדיא" וענה על השאלות האלה:

א. כתוב חוק, המגדיר את היחס שרשרת(_עצם1,_שרשרת,_עצם2), אשר בו המשתנה _שרשרת מסמן את רשימת העצמים, הקושרים בין עצם1 לבין עצם2.

ב. כתוב חוק, המגדיר את היחס מעשים_של(_עצם,_מעשים), אשר בו המשתנה _מעשים מסמן את רשימת המעשים, שמבצע העצם _עצם.

ג. כתוב חוק המגדיר את היחס אורך(_רשימה,_אורך), אשר בו המשתנה _אורך מבטא את מספר האיברים ברשימה _רשימה. הראה באמצעותו, כי רשימת מעשיו של הקדוש-ברוך-הוא היא ארוכה מכולן.

שאלה 3

לפניך הגירסה המורכבת של תכנית The Family (ראה פרק 5):

parent(abraham, isaac).	הורה(אברהם, יצחק).
parent(isaac, jacob).	הורה(יצחק, יעקב).
parent(jacob, reuben).	הורה(יעקב, ראובן).
parent(abraham, ishmael).	הורה(אברהם, ישמעאל).
parent(isaac, esuv).	הורה(יצחק, עשיו).
parent(jacob, shimeon).	הורה(יעקב, שמעון).
parent(sarah, isaac).	הורה(שרה, יצחק).
parent(rivkah, jacob).	הורה(רבקה, יעקב).
parent(leah, reuben).	הורה(לאה, ראובן).
parent(hagar, ishmael).	הורה(הגר, ישמעאל).
parent(rivkah, esuv).	הורה(רבקה, עשיו).
parent(leah, shimeon).	הורה(לאה, שמעון).
male(abraham).	זכר(אברהם).
male(isaac).	זכר(יצחק).
male(esuv).	זכר(עשיו).
male(shimeon).	זכר(שמעון).
male(ishmael).	זכר(ישמעאל).
male(jacob).	זכר(יעקב).
male(reuben).	זכר(ראובן).
female(sarah).	נקבה(שרה).
female(rivkah).	נקבה(רבקה).
female(hagar).	נקבה(הגר).
female(leah).	נקבה(לאה).

א. שנה את מבנה הנתונים שלעיל בהתאם למבנה של נתוני התכנית "קשרי המשפחה המנומקים" של סעיף 3 בחטיבה זו, וענה על שאלה א' של תרגיל 3.

ב. הוסף לתכנית Family Rules של פרק 7 את החוקים, המגדירים את היחסים האלה:
אחים(X,Y)- ל-X ול-Y יש הורה משותף, ו-X שונה מ-Y.
בניו_של(X,Y)- הרשימה Y הינה רשימת בניו של X.

ג. כתוב חוק, המגדיר את היחס צאצאיו_של(X,Y), כאשר X הינו אחד מהגברים של המשפחה התנ"כית, ו-Y היא רשימת צאצאיו ממין זכר.

ד. האם ניתן לשנות את שני החוקים של ancestor(Ancestor,Person) שבסעיף 7.6 (אשר השני שבהם הינו חוק רקורסיבי) כך, שהחוקים האלה יכללו את היחס because בדומה לחוקי התכנית "קשרי המשפחה המנומקים"? נמק את תשובתך.

ה. השווה בין ההסברים, הניתנים באמצעות היחס because, לבין אלה, הניתנים על ידי קונכיית מערכת מומחה באמצעות why ו-how (סעיף 14), ותאר את ההבדלים.

שאלה 4

באיור 39 נמצא החישוב של השאילתה search(a, Y, Rooms) על-פי תכנית 8. השווה בין איורים 12 ו-39. תן הסבר להבדלים שבחישוב.

```
tracing: ¦ ?- search(a,Y,Rooms).
[0] call: search(a,Y81,Rooms82)?
 [1] call: search(a,Y177,[a],Rooms178)?
  [2] call: telephone(a)?
  [2] fail: telephone(a)?
  [2] call: move(a,Link265)?
   [3] call: door(a,B364)?
   [3] exit: door(a,b)?
  [2]·exit: move(a,b)?
  [4] call: member(b,[a])?
  [4] fail: member(b,[a])?
  [4] call: append([a],[b],Visited1269)?
  [4] exit: append([a],[b],[a,b])?
  [4] call: search(b,Y262,[a,b],Rooms264)?
   [5] call: telephone(b)?
   [5] fail: telephone(b)?
   [5] call: move(b,Link766)?
    [6] call: door(b,B865)?
    [6] exit: door(b,d)?
   [5] exit: move(b,d)?
   [7] call: member(d,[a,b])?
   [7] fail: member(d,[a,b])?
   [7] call: append([a,b],[d],Visited1770)?
   [7] exit: append([a,b],[d],[a,b,d])?
   [7] call: search(d,Y763,[a,b,d],Rooms765)?
    [8] call: telephone(d)?
    [8] fail: telephone(d)?
    [8] call: move(d,Link1273)?
     [9] call: door(d,B1372)?
     [9] exit: door(d,h)?
    [8] exit: move(d,h)?
    [10] call: member(h,[a,b,d])?
    [10] fail: member(h,[a,b,d])?
    [10] call: append([a,b,d],[h],Visited1277)?
    [10] exit: append([a,b,d],[h],[a,b,d,h])?
    [10] call: search(h,Y1270,[a,b,d,h],Rooms1272)?.
     [11] call: telephone(h)?
     [11] fail: telephone(h)?
     [11] call: move(h,Link1786)?
      [12] call: door(h,B1885)?
      [12] fail: door(h,_1885)?
      [12] call: door(_1885,h)?
      [12] exit: door(d,h)?
     [11] exit: move(h,d)?
     [13] call: member(d,[a,b,d,h])?
     [13] exit: member(d,[a,b,d,h])?
     [11] redo: move(h,d)?
      [12] redo: door(d,h)?
      [12] fail: door(_1885,h)?
      [12] call: door(h,_1885) ; door(_1885,h)?
      [12] exit: door(h,d) ; door(d,h)?
     [11] exit: move(h,d)?
     [12] call: member(d,[a,b,d,h])?
     [12] exit: member(d,[a,b,d,h])?
     [11] redo: move(h,d)?
      [12] redo: door(h,d) ; door(d,h)?
      [12] fail: door(h,_1885) ; door(_1885,h)?
     [11] fail: move(h,_1786)?
    [10] fail: search(h,_1270,[a,b,d,h],_1272)?
    [10] redo: append([a,b,d],[h],[a,b,d,h])?
    [10] fail: append([a,b,d],[h],_1277)?

    ≈ ≈ ≈

     [11] call: telephone(j)?
     [11] exit: telephone(j)?
    [10] exit: search(j,j,[a,b,e,j],[a,b,e,j])?
   [7] exit: search(e,j,[a,b,e],[a,b,e,j])?
  [4] exit: search(b,j,[a,b],[a,b,e,j])?
 [1] exit: search(a,j,[a],[a,b,e,j])?
[0] exit: search(a,j,[a,b,e,j])?
... in 12 reductions.

Y = j
Rooms = [a,b,e,j]
```

איור 39 – חיפוש הטלפון על-פי תכנית 8

שאלה 5

החלטנו לערוך שינוי מסוים בתכנית 6(ב) של חטיבה זו.
לפניך שלושה איורים של חישוב השאילתה destination(Town) (איורים 40, 41, 42). שניים מהם בוצעו על-פי התכנית ששונתה (לכל איור – שינוי אחר).

א. אתר את החישוב, שנעשה על-פי תכנית 6(ב) לפני השינוי.

ב. קבע על-פי האיורים הנותרים, מהם החוקים אשר שונו.

```
tracing: ¦ ?- destination(Town).
[0] call: destination(Town80)?
 [1] call: has_chance(situation(Town171,Employment173))?
  [2] call: Town255 = b?
  [2] exit: b = b?
 [1] exit: has_chance(situation(b,has))?
[0] exit: destination(b)?
... in 2 reductions.

Town = b ;
[0] redo: destination(b)?
 [1] redo: has_chance(situation(b,has))?
  [2] redo: b = b?
  [2] fail: _255 = b?
  [2] call: _255 = c?
  [2] exit: c = c?
 [1] exit: has_chance(situation(c,has))?
[0] exit: destination(c)?
... in 2 reductions.

Town = c ;
[0] redo: destination(c)?
 [1] redo: has_chance(situation(c,has))?
  [2] redo: c = c?
  [2] fail: _255 = c?
  [2] call: _255 = f?
  [2] exit: f = f?
 [1] exit: has_chance(situation(f,has))?
[0] exit: destination(f)?
... in 2 reductions.

Town = f ;
[0] redo: destination(f)?
 [1] redo: has_chance(situation(f,has))?
  [2] redo: f = f?
  [2] fail: _255 = f?
  [2] call: _255 = c ; _255 = f?
  [2] exit: c = c ; c = f?
 [1] exit: has_chance(situation(c,has))?
[0] exit: destination(c)?
... in 2 reductions.

Town = c ;
[0] redo: destination(c)?
 [1] redo: has_chance(situation(c,has))?
  [2] redo: c = c ; c = f?
  [2] exit: f = c ; f = f?
 [1] exit: has_chance(situation(f,has))?
[0] exit: destination(f)?
... in 2 reductions.

Town = f ;
[0] redo: destination(f)?
 [1] redo: has_chance(situation(f,has))?
  [2] redo: f = c ; f = f?
  [2] fail: _255 = c ; _255 = f?
  [2] call: _255 = b ; _255 = c ; _255 = f?
  [2] exit: b = b ; b = c ; b = f?
 [1] exit: has_chance(situation(b,has))?
[0] exit: destination(b)?
... in 2 reductions.

Town = b ;
[0] redo: destination(b)?
 [1] redo: has_chance(situation(b,has))?
  [2] redo: b = b ; b = c ; b = f?
  [2] exit: c = b ; c = c ; c = f?
 [1] exit: has_chance(situation(c,has))?
[0] exit: destination(c)?
... in 2 reductions.

Town = c ;
[0] redo: destination(c)?
 [1] redo: has_chance(situation(c,has))?
  [2] redo: c = b ; c = c ; c = f?
  [2] exit: f = b ; f = c ; f = f?
 [1] exit: has_chance(situation(f,has))?
[0] exit: destination(f)?
... in 2 reductions.

Town = f
```

איור 40 – destination(Town), גירסה ראשונה

```
tracing: | ?- destination(Town).
[0] call: destination(Town80)?
 [1] call: has_chance(situation(Town171,X173))?
  [2] call: situation(Town171,X173) = situation(T256,has)?
  [2] exit: situation(T256,has) = situation(T256,has)?
 [1] exit: has_chance(situation(T256,has))?
[0] exit: destination(T256)?
... in 2 reductions.

Town = T256 ;
[0] redo: destination(T256)?
 [1] redo: has_chance(situation(T256,has))?
  [2] redo: situation(T256,has) = situation(T256,has)?
  [2] fail: situation(_171,_173) = situation(T256,has)?
  [2] call: move(situation(_171,_173),Situation2255)?
   [3] call: com(Town1346,Town2349)?
   [3] exit: com(a,b)?
  [2] exit: move(situation(a,S1347),situation(b,S2350))?
  [4] call: has_chance(situation(b,S2350))?
   [5] call: situation(b,S2350) = situation(T601,has)?
   [5] exit: situation(b,has) = situation(b,has)?
  [4] exit: has_chance(situation(b,has))?
 [1] exit: has_chance(situation(a,S1347))?
[0] exit: destination(a)?
... in 5 reductions.

Town = a ;
[0] redo: destination(a)?
 [1] redo: has_chance(situation(a,S1347))?
  [4] redo: has_chance(situation(b,has))?
   [5] redo: situation(b,has) = situation(b,has)?
   [5] fail: situation(b,_350) = situation(_601,has)?
   [5] call: move(situation(b,_350),Situation2600)?
    [6] call: com(b,Town2694)?
    [6] exit: com(b,e)?
   [5] exit: move(situation(b,S1692),situation(e,S2695))?
   [7] call: has_chance(situation(e,S2695))?
    [8] call: situation(e,S2695) = situation(T946,has)?
    [8] exit: situation(e,has) = situation(e,has)?
   [7] exit: has_chance(situation(e,has))?
  [4] exit: has_chance(situation(b,S1692))?
 [1] exit: has_chance(situation(a,S1347))?
[0] exit: destination(a)?
... in 8 reductions.
```

איור 41 – destination(Town), גירסה שניה

```
Town = a ;
[0] redo: destination(a)?
 [1] redo: has_chance(situation(a,S1347))?
  [4] redo: has_chance(situation(b,S1692))?
   [7] redo: has_chance(situation(e,has))?
    [8] redo: situation(e,has) = situation(e,has)?
    [8] fail: situation(e,_695) = situation(_946,has)?
    [8] call: move(situation(e,_695),Situation2945)?
     [9] call: com(e,Town21039)?
     [9] fail: com(e,_1039)?
    [8] fail: move(situation(e,_695),_945)?
   [7] fail: has_chance(situation(e,_695))?
   [5] redo: move(situation(b,S1692),situation(e,_695))?
    [6] redo: com(b,e)?
    [6] fail: com(b,_694)?
   [5] fail: move(situation(b,_350),_600)?
  [4] fail: has_chance(situation(b,_350))?
  [2] redo: move(situation(a,S1347),situation(b,_350))?
   [3] redo: com(a,b)?
   [3] exit: com(b,e)?
  [2] exit: move(situation(b,S1347),situation(e,_350))?
  [4] call: has_chance(situation(e,_350))?
   [5] call: situation(e,_350) = situation(T601,has)?
   [5] exit: situation(e,has) = situation(e,has)?
  [4] exit: has_chance(situation(e,has))?
 [1] exit: has_chance(situation(b,S1347))?
[0] exit: destination(b)?
... in 5 reductions.

Town = b ;
[0] redo: destination(b)?
 [1] redo: has_chance(situation(b,S1347))?
  [4] redo: has_chance(situation(e,has))?
   [5] redo: situation(e,has) = situation(e,has)?
   [5] fail: situation(e,_350) = situation(_601,has)?
   [5] call: move(situation(e,_350),Situation2600)?
    [6] call: com(e,Town2694)?
    [6] fail: com(e,_694)?
   [5] fail: move(situation(e,_350),_600)?
  [4] fail: has_chance(situation(e,_350))?
  [2] redo: move(situation(b,S1347),situation(e,_350))?
   [3] redo: com(b,e)?
   [3] exit: com(d,c)?
  [2] exit: move(situation(d,S1347),situation(c,_350))?
  [4] call: has_chance(situation(c,_350))?
   [5] call: situation(c,_350) = situation(T601,has)?
   [5] exit: situation(c,has) = situation(c,has)?
  [4] exit: has_chance(situation(c,has))?
 [1] exit: has_chance(situation(d,S1347))?
[0] exit: destination(d)?
... in 5 reductions.

Town = d y
```

```
tracing: ¦ ?- dest(Town).
[0] call: dest(Town80)?
 [1] call: has_chance(situation(Town171,Employment173))?
  [2] call: situation(Town255,has)?
  [2] exit: situation(b,has)?
 [1] exit: has_chance(situation(b,has))?
[0] exit: dest(b)?
... in 3 reductions.

Town = b ;
[0] redo: dest(b)?
 [1] redo: has_chance(situation(b,has))?
  [2] redo: situation(b,has)?
  [2] exit: situation(f,has)?
 [1] exit: has_chance(situation(f,has))?
[0] exit: dest(f)?
... in 3 reductions.

Town = f ;
[0] redo: dest(f)?
 [1] redo: has_chance(situation(f,has))?
  [2] redo: situation(f,has)?
  [2] exit: situation(c,has)?
 [1] exit: has_chance(situation(c,has))?
[0] exit: dest(c)?
... in 3 reductions.

Town = c ;
[0] redo: dest(c)?
 [1] redo: has_chance(situation(c,has))?
  [2] redo: situation(c,has)?
  [2] fail: situation(_255,has)?
  [2] call: move(situation(_171,_173),Situation2255)?
   [3] call: com(Town1346,Town2349)?
   [3] exit: com(a,b)?
  [2] exit: move(situation(a,Employ1347),situation(b,Employ2350))?
  [4] call: has_chance(situation(b,Employ2350))?
   [5] call: situation(b,has)?
   [5] exit: situation(b,has)?
  [4] exit: has_chance(situation(b,has))?
 [1] exit: has_chance(situation(a,Employ1347))?
[0] exit: dest(a)?
... in 6 reductions.
```

```
Town = a ;
[0] redo: dest(a)?
 [1] redo: has_chance(situation(a,Employ1347))?
  [4] redo: has_chance(situation(b,has))?
   [5] redo: situation(b,has)?
   [5] fail: situation(b,has)?
   [5] call: move(situation(b,_350),Situation2600)?
    [6] call: com(b,Town2694)?
    [6] exit: com(b,e)?
   [5] exit: move(situation(b,Employ1692),situation(e,Employ2695))?
   [7] call: has_chance(situation(e,Employ2695))?
    [8] call: situation(e,has)?
    [8] fail: situation(e,has)?
    [8] call: move(situation(e,_695),Situation2945)?
     [9] call: com(e,Town21039)?
     [9] fail: com(e,_1039)?
    [8] fail: move(situation(e,_695),_945)?
   [7] fail: has_chance(situation(e,_695))?
   [5] redo: move(situation(b,Employ1692),situation(e,_695))?
    [6] redo: com(b,e)?
    [6] fail: com(b,_694)?
   [5] fail: move(situation(b,_350),_600)?
  [4] fail: has_chance(situation(b,_350))?
  [2] redo: move(situation(a,Employ1347),situation(b,_350))?
   [3] redo: com(a,b)?
   [3] exit: com(b,e)?
  [2] exit: move(situation(b,Employ1347),situation(e,_350))?
  [4] call: has_chance(situation(e,_350))?
   [5] call: situation(e,has)?
   [5] fail: situation(e,has)?
   [5] call: move(situation(e,_350),Situation2600)?
    [6] call: com(e,Town2694)?
    [6] fail: com(e,_694)?
   [5] fail: move(situation(e,_350),_600)?
  [4] fail: has_chance(situation(e,_350))?
  [2] redo: move(situation(b,Employ1347),situation(e,_350))?
   [3] redo: com(b,e)?
   [3] exit: com(d,c)?
  [2] exit: move(situation(d,Employ1347),situation(c,_350))?
  [4] call: has_chance(situation(c,_350))?
   [5] call: situation(c,has)?
   [5] exit: situation(c,has)?
  [4] exit: has_chance(situation(c,has))?
 [1] exit: has_chance(situation(d,Employ1347))?
[0] exit: dest(d)?
... in 6 reductions.

Town = d
```

איור 42 – destination(Town), גירסה שלישית

שאלה 6

א. תלמידי כיתה י"א החליטו לערוך טיול באתרים היסטוריים שבין תל אביב לבין חיפה. בחר את היעדים, ותאר באמצעות תכנית לוגית את מסלולי הטיול האפשריים.

ב. תלמידי הכיתה המקבילה החליטו לערוך תחרות ריצה בין תל אביב לבין חיפה. תאר באמצעות תכנית לוגית את מסלולי הריצה האפשריים וציין את אורכו של כל מסלול.

ג. הנהלת בית הספר החליטה לקראת חג החנוכה לערוך טיול בעקבות המכבים. בנה תכנית לוגית, אשר תסייע בבחירת המסלולים האפשריים. בנוסף למשמעותם ההסטורי של היעדים, ציין את המרחקים שביניהם, ואת אורכם של המסלולים.

הערה: בתכנית אשר תבנה, אתה רשאי להחליט על מקום התחלת הטיול ועל מקום סיומו.

שאלה 7

החלטנו לערוך שינוי בפרוצדורה solve5 של סעיף 9. עיין באיור 43 ותאר את השינוי ואת תוצאותיו.

```
tracing: ¦ ?- solve5(a,History,Moves).
[0] call: solve5(a,History81,Moves82)?
 [1] call: final_state(a)?
 [1] fail: final_state(a)?
 [1] call: move(a,Move179,Link181)?
 [1] exit: move(a,move(a,b),b)?
 [2] call: member(b,History177)?
 [2] exit: member(b,[b¦_409])?
 [1] redo: move(a,move(a,b),b)?
 [1] exit: move(a,move(a,c),c)?
 [2] call: member(c,_177)?
 [2] exit: member(c,[c¦_409])?
 [1] redo: move(a,move(a,c),c)?
 [1] fail: move(a,_179,_181)?
[0] fail: solve5(a,_81,_82)?

no
```

איור 43 – שינוי של הפרוצדורה solve5

שאלה 8

הפונקציות הבאות מוגדרות עבור כל זוג של מספרים טבעיים x ו-y. תאר כל אחת מהן באמצעות תכנית לוגית.

א $$f(x,y) = \begin{cases} 1 & x \le y \\ 0 & x > y \end{cases}$$

ב. $$f(x,y) = \begin{cases} x! & x \le y \\ y! & x > y \end{cases}$$

ג. $f(x,y) = \min(x!, y!)$

ד. $f(x,y) = \min(2x+y, x+2y)$

ה.

$$f(x,y) = \begin{cases} y & x=0 \\ f(x-1, 0) & y=0 \\ f(x-1, f(x, y-1)) & x>0, y>0 \end{cases}$$

ו. תאר את הפונקציה האחרונה באמצעות תכנית לוגית, שהמספרים הטבעיים בה מיוצגים כצאצאים של 0: s(0), s(s(0)), וכן הלאה. בדוק באמצעות התכנית הזאת מהו הערך של הפונקציה עבור זוג כלשהו של מספרים טבעיים.

שאלה 9

כתוב מערכת מומחה בנושא התפתחות בעלי החיים. היעזר בתכניות שבסעיפים 12, 13, 14.

שאלה 10

א. הוסף לקונכייה של מערכת מומחה שבסעיף 14 פרוצדורה, אשר באמצעותה ניתן למחוק את התשובות של המשתמש (בדומה לפרוצדורה clear_facts שבסעיף 13).

ב. הוסף לפרוצדורה solve של הקונכיה חוקים, אשר באמצעותם ניתן לחשב את המטרה not Goal.

ג. בנה מערכת מומחה בנושא המוכר לך. פנה אל חבר שמכיר את הנושא, ובקשו להעריך את המערכת מנקודת ראות של המשתמש. שפר את המערכת בהתאם להערותיו. הסבר לו מה הן המגבלות של השימוש בה, במידה ולא ניתן לשפר אותה בהתאם להערותיו.

נספחים

נספח א׳ — סביבת עבודה WISDOM-PROLOG

נספח ב׳ — אופרטורים

נספח ג׳ — סביבת עבודה TI - PROLOG

נספח ד׳ — הפרוצדורה append בפסקל ובפרולוג

נספח ה׳ — סביבת עבודה TURBO-PROLOG

נספח א׳

סביבת עבודה WISDOM-PROLOG

נמסר באדיבות: Maxbyte Computer Software
ת.ד. 7925, ירושלים.

1. תיאור כללי של תוכנת WISDOM

התוכנה WISDOM-PROLOG (בקיצור-WISDOM) פותחה על ידי מולי ספרא (מכון וייצמן למדע, רחובות) ומוטי גולדברג (מקסבייט תוכנת מחשבים, ירושלים). השם WISDOM (שמשמעו באנגלית – בינה, תבונה) נגזר מצירוף המילים:
Weizmann Institute of Science, Department Of Mathematics

המערכת של WISDOM בנויה מגרעין (בשפת C) ומקונכיה (בשפת פרולוג). הגרעין מגדיר את מנגנון ההסקה ואת המישקים למכונה המארחת. הוא "עטוף" בקונכיה, המכילה דקדוק של שפה, פרדיקטים של המערכת ומישקים למשתמש. הקונכיה נכתבה בפרולוג, ועל כן המשתמש יכול לשנותה כדי ליצור סביבת עבודה בהתאם לדרישותיו. השינויים יכולים להתבצע בשתי רמות:

א. שימוש ב"יציאות" של הקונכיה המקורית לפרדיקטים, שהוגדרו על ידי המשתמש.
ב. שינוי של פרדיקטי המערכת.

להבדיל מקונכיה של מערכת מומחה, אשר נדונה בחטיבה הרביעית, הקונכיה שעוטפת את הגרעין מכילה בנוסף למפרש-על, את כלל סביבת המשתמש.

תוכנת WISDOM משמשת שתי מטרות עיקריות:
1. להוראה של תכנות לוגי.
2. ליישומים מתוחכמים, הנבנים על פי עקרון של תכנות בוחן והמשתמשים במנגנון הסקה לוגית (בניית אבות-טיפוס, מערכות מומחה, עיבוד שפות טבעיות וכדומה).

המערכת מיושמת על מגוון רחב של מחשבים ומערכות ההפעלה. כדי להגיע לרמת תאימות גבוהה זו, המפתחים החליטו:

– לנקוט בגישה מודולרית, שתאפשר העברת WISDOM ממחשב או ממערכת הפעלה אחד למשנהו על ידי החלפת מודולים מעטים בלבד.
– להשתמש בשפת C ולהגביל את השימוש בשפת סף (assembly).
– במידת האפשר, לכתוב WISDOM בפרולוג – כלומר, המערכת מסוגלת לבנות את עצמה מגרעין מצומצם.

להלן רשימת הגירסאות והמכונות המארחות:

מערכת הפעלה	מחשב	גירסה
MS-DOS	IBM-PC (256 KB)	Easy-Wisdom למתחילים, עברי+אנגלי
MS-DOS	IBM-PC (384 KB)	Wisdom-PC עם חלונות
Apple-DOS	MACINTOSH	Wisdom-MAC
S-8000	ZILOG	Wisdom-ZIL
UNIX	AT & T	Wisdom-ATT
UNIX	MOTOROLA	Wisdom-M
UNIX	APOLLO	Wisdom-AP
UNIX	SUN	Wisdom-SUN
UNIX	CCI	Wisdom-CCI
VMS	VAX	Wisdom-VAX-VMS
UNIX	VAX	Wisdom-VAX-U

התחביר של WISDOM הינו מערכת-על של תקן "אדינבורו". בכל הגירסאות, פרט ל-Easy, ניתנת תמיכה מלאה במספרים עם נקודה צפה.
חבילת Easy, התומכת בדקדוק עברי ואנגלי, מכילה מפרש, קובץ מידע ומערכת ניפוי.
חבילות אחרות (פרט ל-Easy) מכילות:

* מפרש – קובץ הפעלה.
* קבצי ידע (state files).
* קובץ מקור של מערכת הניפוי, שנכתב בפרולוג (כולל spy points וניפוי תמידי).
* חלקים נרחבים של קונכיית הפרולוג.
* ספריות שונות (כולל דוגמה ל-Definite Clause Grammar).
* מודול מישק לשפות: C, אסמבלר, פסקל, פורטרן.
* תיעוד למשתמש – user manual.

בנוסף חבילת Wisdom-PC מכילה:

* תכנית ניהול חלונות + ספריה למתכנת.
* מערכת לבניה והצגה של חלונות עזרה.
* תכנית פרולוג לעורך (editor).
* דוגמאות למישק עם MS-DOS ו-BIOS interrupts.

המשתמש של Wisdom-PC יכול לשלוט במערכת הקולית (sound).

2. מערכת הניפוי – tracer, debugger

התכנית של פרולוג בדרך כלל אינה מפרטת את צעדי הביצוע, ותהליך הרצתה (כלומר, חישוב השאילתה) מוסתר מעיני המשתמש. תהליך זה כולל נסיגה (backtracking), ועל כן מהלכיו קשים לחיזוי. בשל כך המערכת של WISDOM מכילה תוכנת ניפוי (debugging), המאפשרת לעקוב (trace) אחרי ריצת התכנית.
מודל מיוחד הוגדר כדי לתאר את מהלכי ביצוע של תכנית: קופסה בעלת 4 פתחים של כניסות/יציאות למעבר תכניות. בהתאם לפתח, אשר דרכו עברה תכנית מסוימת, ניתן לראות מתי מטרה (goal) הצליחה, נסוגה או נכשלה.

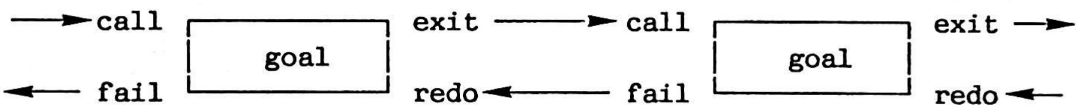
ארבעת הפתחים הם:

CALL – פתח כניסה ראשונה לכל goal. מספר הכניסה מאפשר להבחין בין כניסות שונות של אותו goal בארועים **שונים** שלו בתכנית.

EXIT – פתח יציאה לאחר הצלחה.

REDO – פתח כניסה מחדש של התכנית ל-goal בנסיון נוסף (נסיגה).

FAIL – פתח יציאה לאחר כשלון.



בנוסף לאופציות העקיבה (trace) הסטנדרטיות, המערכת מאפשרת הגדרת עקיבה מתמדת, הגדרת נקודות עקיבה ספציפיות (spy points) ואף אפשרות יציאה זמנית למערכת WISDOM הסטנדרטית, כדי לבצע שינויים "בזמן אמיתי", לחזור למערכת הניפוי ולבדוק את השפעת השינוי על התנהגות התכנית.

תכנית ניפוי כתובה בפרולוג ומהווה חלק מחבילת התוכנה הנמסרת למשתמש, כדי שיוכל ללמוד אותה, או אף לבצע בה שינויים כרצונו.

3. עבודה מול תכניות ומודולים חיצוניים

היכולת לתקשר עם תוכנות אחרות הינה אחת התכונות החשובות שהמתכנת מחפש בכל מערכת או שפה. WISDOM מאפשרת זאת בשלושה אופנים שונים:

א. יציאה זמנית למערכת ההפעלה, תוך כדי העלאת עותק נוסף של מערכת ההפעלה על-גבי מערכת WISDOM.

ב. הפעלת תכנית חיצונית מתוך WISDOM. במקרה זה עותק נוסף של מערכת ההפעלה אינו מועלה לזכרון המחשב (כלומר נחסך זיכרון), אך עם זאת לא ניתן לבצע את כל הפעולות שהתאפשרו במקרה הקודם (למשל, אי-אפשר לבצע פקודת DIR או COPY תחת MS-DOS).

ג. חיבור מודולים נוספים ל-WISDOM- תוספת גרעין למערכת. המשתמש יכול לחבר WISDOM עם בסיס נתונים, אם כ-front-end לאחר שהגדיר בפרולוג SQL מתואם, ואם בתור מנגנון ההסקה.
לצורך זה הוגדרו ב-WISDOM מספר פרדיקטים יחודיים, המאפשרים למודולים נוספים להשתתף עם המערכת המקורית באיזורי זיכרון ולהקרא/להתבצע מתוך WISDOM.

כדי לאפשר את החיבור של מודולי המשתמש ל-WISDOM, המודולים השונים מצורפים בגרסתם המהודרת (compiled code). שיטה זו נבחרה כיון שהיא היחידה המשותפת לכל המחשבים שבהם פועלת WISDOM: תחת MS-DOS ניתן להתקשר דרך מערכת ה-interrupts, תחת VMS באמצעות dynamic link, תחת UNIX קיימות שיטות שונות בהתאם לגירסה. אף לא אחת משיטות אלו קיימת במערכות אחרות.

להלן מספר דוגמאות של תכניות/מודולים זרים.

א. ביצוע פקודה של מערכת הפעלה:

```
system('dir *.pl /p').
```

ב. הפעלת תכנית:

```
fork_exec(edlin, 'my-file.pl', RetVal).
```

ג. תחת SWM (מערכת ניהול חלונות של WISDOM, ראה בהמשך), ניתן לקשור את הפקודות למקשים פונקציונליים:

```
bind_fkey(3, 1, 'dir *.pl /p', _, _).

bind_fkey(4, 2, edlin, 'my-file.pl', RetVal).
```

ואז הקשה על מקש F3 תבצע את א' והקשה על F4 את ב'.

ד. קריאה לרוטינות המשתמש מתוך Wisdom:

```
usr_func(Func, ArgList, RetVal).
```

כאשר: Func הוא מספר הפונקציה המבוקשת.
ArgList היא רשימה של הארגומנטים להעברה לפונקציה.
RetVal הוא הערך המוחזר ע"י הפונקציה.

4. טיפול במערכים

תוכנת WISDOM מאפשרת לנהל מערכים (arrays) בפרולוג. התכנית המנהלת (תכנית א') מבוססת על הפרדיקט המאפשר הקצאת אזור זכרון (allocation) לצורך שיתוף נתונים עם מודולים חיצוניים. הקוד המפורט כאן מתייחס למערכים של סוגי נתונים מוגדרים. אולם, ברור כי הדבר אינו הכרחי ואכן, בקובץ המלא, נמצאת ההדרכה המתאימה כדי להרחיב את התכנית כך שתדע לטפל גם באלמנטים מורכבים שונים במערך יחיד.

הפונקציות השונות בתכנית:

א. הגדר מערך עם שם, סוג ומימדים.
ב. שחרר מערך.
ג. קבע ערך לתא במערך.
ד. קבל ערך מתוך תא במערך.

השלבים בהגדרת מערך:

א. מצא את נפח הזכרון הדרוש כדי לאחסן את הנתונים במערך.
ב. דרוש את הזכרון.
ג. שמור את הגדרת המערכת להמשך עבודה.

השלבים בשחרור מערך:

א. שחרר את הזכרון שנדרש למערך.
ב. מחק את הגדרת המערך.

גישה לתא במערך:

א. חשב את מיקומו.
ב. קרא/עדכן את תוכנו.

```
/* Declare an array.
   The code here does not permit the declaration of two arrays by the same
   name. You could change the first goal to
             not(array(Name, _, Type, Dimension_list))
   to allow a more Prolog-like syntax, where several arrays with the same name
   but different data-type and/or dimensions could be declared.
*/
define_array(Name, Dtype, Dim_list) :-
        not(array(Name, _, _, _)),                  % unused name for array
        $unit(Dtype, Unit),                         % get data-type unit value
        $total_size(Dim_list, Tsize), Tsize \= 0,    % calculate space
        buffer(Array, L1),                           % allocate the array
        assert(array(Name, Array, Dtype, Dim_list)).  % store declaration

/* Free the space of an array.
   If you allowed multiple arrays with the same name, the definition should be
        array(Name, Array, Dtype, Dim_list),
        buffer(Array, _),
        retract(array(Name, Array, Dtype, Dim_list)).
*/
delete_array(Name) :-
        array(Name, Array, _, _),          % get array pointer
        buffer(Array, _),                  % deallocate its buffer
        retract(array(Name, _, _)).        % erase its stored declaration

/* set/get an array-entry.
   set - when Val is instantiated.
   get - when Val is a variable.
   If you allowed multiple arrays with the same name & Dlist, the head of the
   clause should be defined as
              element(Name, Type, Indices, Val) :-
*/
element(Name, Indices, Val) :-
        array(Name, Array, Type, Dlist),        % find declaration
        $unit(Type, Unit),
        $ndx(Indices, Dlist, _, OneIndex), Place is OneIndex*Unit,
        rwbuffer(Array, Place, Val, Type).

/* define unit-values (in byte length) for the different data-types
        (PC-version - for MAC/UNIX/VMS versions you should set i & u values
        to 4).
*/
$unit(c, 1).     % character              Values for PC-version
$unit(i, 2).     % integer                (±32767)
$unit(u, 2).     % unsigned integer       (0 - +65536)
$unit(si, 2).    % short integer
$unit(su, 2).    % short unsigned integer
$unit(l, 4).     % long  (WP's integer)
$unit(f, 4).     % float
$unit(p, 4).     % pointer - to use with atom_ref/2

$total_size([Dim|DimList], Tsize) :-
        DimList \= [], $total_size(DimList, Size), Tsize is Dim*Size.
$total_size([Dim], Dim).

/* convert a (possible) multi-dimensional entry to its equivalent
   single-dimension entry. The array entries are 0 to N-1, where N is
   the dimension's number of entries.
   No boundary-checking is done.
   If you allowed multiple arrays with the same name & Dlist, you should
   add the test for it.
*/
$ndx([Ndx|Ndxs], [Dim|Dims], Dim*NextDims, Val) :-
        Ndxs \= [],
        $ndx(Ndxs, Dims, NextDims, V),
        Val = Ndx*NextDims+V.
$ndx([Ndx], [Dim], Dim, Ndx).

/* define an operational way to handle array-elements
*/
:- op(930, fx, set).
:- op(925, xfy, to).

/* The following definition is valid for both setting an array-element
   to a given value, (e.g. set array1[3,4] to 18) as well as setting a
   variable from an array-entry. (e.g. set array[3,4] to Val)
*/
set Array_entry to Val :-
        Array_entry =.. [Name, Indices], element(Name, Indices, Val).

/* for clarity, you may define the following for setting a variable from an
   array-entry
*/
:- op(925, xfy, from).
set Val from Array_entry :-
        set Array_entry to Val.
```

5. מערכת ניהול חלונות

זה מכבר מקובל בתכניות PC, כי תכנית אשר תומכת בחלונות לצורך תצוגה, הינה תכנית "ידידותית" (user friendly). החלונות הינם כלי יעיל להפרדה של אלמנטים שונים לצורך תצוגה על גבי מסך פיסי יחיד. ברמת התוכנה, המתכנת יכול להגדיר לאיזה מהחלונות ברצונו לכתוב, מאיזה לקרוא והיכן.
WISDOM מאפשר שימוש בשני אופני תצוגה:

- הצורה הסטנדרטית בה מוצגות השורות בזו אחר זו על גבי מסך אחד.
- תחת מערכת SWM (Simplified Window Manager) – מערכת ניהול החלונות של WISDOM.

SWM הינה מערכת ניהול חלונות פשוטה להפעלה שנבנתה על בסיס מערכת WM רבת עוצמה. המערכת המקורית, WM, מאפשרת למשתמש לנהל מספר אובייקטים:

(VS) מסכים בפועל	–	"דפים" בגודל המוגדר ע"י המשתמש ומהווים אזורי החסן לנתונים המיועדים לתצוגה.
(VP) חלונות תצוגה	–	"פתחים" על המסך הפיסי דרכם יוצג החלק הרצוי מתוך המסך בפועל. ע"פ הגדרתו, גודלו המקסימלי של חלון תצוגה הוא כגודל המסך הפיסי. מספר חלונות תצוגה יכולים להתייחס ל-VS יחיד.
(FR) מסגרות	–	תוחמים לחלונות התצוגה.
(NM) שמות	–	שמות לחלונות התצוגה.
(CRS) סמנים	–	לכל מסך בפועל ולכל חלון תצוגה יש סמן נפרד.
(SV) תצוגת-על	–	מספר חלונות תצוגה יכולים להקשר במסגרת תצוגת-על, כך שתזוזת אחד מהם תגרור תזוזה בהתאם של האחרים.
(VPL) רשימת חלונות	–	רשימה לניהול סדר הופעת חלונות התצוגה.

לכל אחד מהאובייקטים מוגדרות פעולות מותרות כגון הזזה, שינוי גודל, כתיבה, קריאה ועוד. הספריה בשלמותה מסופקת עם מערכת WISDOM כדי להקל על המשתמש המעונין ביצירת תכניות המשתמשות במערכת זו.
בעוד מערכת WM מספקת למשתמש מנוסה כלים רבי עוצמה, אין ספק כי מורכבותה מהווה מכשלה לרוב המשתמשים – ברצונם להשתמש בכלי פשוט ונוח לתפעול, גם אם במחיר הפסד חלק מאפשרויות הביצוע. לצורך זה נבנתה מערכת SWM.
ה-SWM מגדירה אובייקט יחיד – CO - הכולל בתוכו מסך בפועל, חלון תצוגה (אחד לכל VS), מסגרת, שם וסמן. באופן כזה צומצם מספר הפרדיקטים שהמשתמש צריך לזכור על מנת לשלוט במערכת החלונות. בנוסף, תומכת המערכת בעורך (editor) פשוט בכל חלון, כולל העברת תמליל מחלון למשנהו. עם זאת, המערכת חזקה דיה כדי לבנות בעזרתה מעבד תמלילים מלא!

בצד התמיכה בחלונות המוגדרים ע"י המשתמש, תומכת המערכת במספר חלונות WISDOM המוגדרים מראש.
שני שיקולים הנחו את בנית מישק המערכות WISDOM – (SWM): Simplified Window Manager

א. בצד הרצון לתת בידי המשתמש החפץ בכך, שליטה מירבית על ניהול התצוגה, קיימות המגבלות הטכניות של ניהול ערוצי התקשורת היחידים של WISDOM עם המסוף.

כתוצאה, נוצרים חלונות מערכת WISDOM בקוד הגרעין. הם לא ניתנים לשינוי (ולכן לסגירה בטעות), אך כל פעולות הקלט/פלט מקודדות בפרולוג בקוד הנמסר לידי המשתמש.

ב. אין הבדל בפרולוג בין סוגי המשתנים השונים, כאשר מטרתה היחידה של המערכת היא הוכחת ה-goal הראשוני שלה, ה-shell. אם כך, אילו חלונות נפרדים יופיעו על המסך? הפתרון היה בהחלטה שקבעה, כי אלו הן הפעולות של המערכת, הנפרדות ביותר מבחינת ההבנה שלנו:
- דו שיח רגיל משתמש-מערכת.
- אימות נתונים (assertion).
- עקיבה (trace).

נסכם ונאמר: מבחינת המערכת יש פעולה אחת בלבד: הוכחת ה-goal. המשתמש רואה את הפעולה הזאת כמורכבת ממספר פעולות, ועל-פי ראיה זו נבנו החלונות.

בנוסף לתמיכתה במערכת החלונות, מערכת SWM מאפשרת למשתמש להגדיר מקשים פונקציונליים לצורך ביצוע פעולות מוגדרות הקשורות בתכניות חיצוניות.

שתי גירסאות של תכנית ELIZA (ראה תכנית ב') מדגימות הסבת תכנית כדי שתתמוך בעבודה עם חלונות. נציין כי התכנית המוסבת לעבודה עם חלונות, תרוץ גם ללא מערכת החלונות.

```
C:\PROLOG>wisdom -d100 -s4 stdl
Wisdom(tm) Prolog interpreter version 2.20
Copyright (C) M. Safra & M.Goldberg(MAXBYTE), Israel
Copyright (C) MIT Press (U.S.A)

es
 ?-
```

```
<<<Input Active>>> [F1 - Drag] [F2 - Resize] [F9 - CopyActive] [F10 - Next]
----------------------[user] Window----------------------

-------Dialog Window-------        -------Trace Window-------
yes
 ?-
```

איור א' – מסך פתיחה של WISDOM: א. בלי חלונות,
ב. עם חלונות

```prolog
eliza :- read_sentence(Input), eliza(Input), !.

eliza([bye]) :-

   writeln(['Goodbye. I hope I have helped you']).

eliza(Input) :-
   replace_words(Input_, Input),
   pattern(Stimulus,Response),
   match(Stimulus,Table,Input),
   match(Response,Table,Output),

   reply(Output),

   read_sentence(Input1),
   eliza(Input1).
```

חלק א' - בלי חלונות

```prolog
match([N|Pattern],Table,Target) :-
   integer(N),
   lookup(N,Table,LeftTarget),
   append(LeftTarget,RightTarget,Target),
   match(Pattern,Table,RightTarget).
match([Word|Pattern],Table,[Word|Target]) :-
   atom(Word),
   match(Pattern,Table,Target).
match([],Table,[]).

pattern([i,am,1],[how,long,have,you,been,1,?]).
pattern([1,you,2,me],[what,makes,you,think,i,2,you,?]).
pattern([i,like,1],[does,anyone,else,in,your,family,like,1,?]).
pattern([i,feel,1],[do,you,often,feel,that,way,?]).
pattern([1,X,2],[can,you,tell,me,more,about,X]) :-
      important(X).
pattern([1],[please,go,on]).
    important(father).                      important(mother).
    important(sister).                      important(brother).

lookup(X,[(X,V)|XVs],V).
lookup(X,[(X1,V1)|XVs],V) :- X \= X1, lookup(X,XVs,V).

replace_words([Word1|L],[Word2|L1]) :-
      subst(Word1, Word2), replace_words(L,L1). % , !.
replace_words(L,L1) :- subst(L,L1). % , !.
subst(my, your).
subst(your, my).
subst(P,P).

writeln(X) :- reply(X), nl.
reply([Head|Tail]) :- write(Head), write(' '), reply(Tail).
reply([]) :- nl.
read_sentence(Xs) :- read(Xs).
```

חלק ב' - משותף

```prolog
eliza :- eliza_won(I,O,CW),
      ((read_sentence(Input), eliza(Input,I,O)) ; true), !,
      eliza_woff(I,O,CW).
eliza_won(0,0,CW) :- not(wmison).
eliza_won(I,O,CW) :- wmcreate(2,2,30,20,38,30,1,'PATIENT',I),
      wmcreate(2,40,30,20,39,30,1,'ANALYSIST',O),
      wmcw(CW), wmno(I).
eliza_woff(0,0,_).
eliza_woff(I,O,CW) :- wmdelete(I), wmdelete(O), wmno(CW).

eliza([bye],_,O) :-
      wmno(O),
      writeln(['Goodbye. I hope I have helped you']), wait.
wait :- not(wmison).
wait :- getch(_).
wmison :- wmcw(CW), wmno(2), wmcw(C), C=2, wmno(CW).

eliza(Input_,I,O) :-
   replace_words(Input_, Input),
   pattern(Stimulus,Response),
   match(Stimulus,Table,Input),
   match(Response,Table,Output),
   wmno(O),
   reply(Output),
   wmno(I),
   read_sentence(Input1),
   eliza(Input1,I,O).
```

חלק ג' - עם חלונות

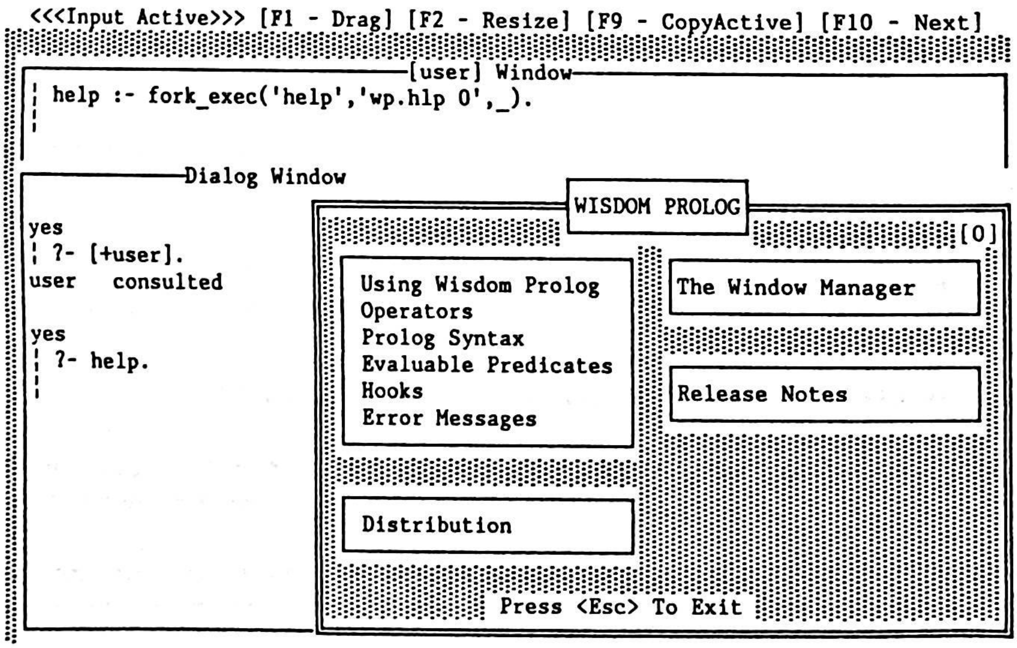
תכנית ב' – שתי גירסאות של ELIZA: עם ובלי חלונות

6. מערכת ניהול ספריות מסכי עזרה

מערכת העזרה מאפשרת למשתמש לבנות מסכי עזרה (כולל תמיכה בצבע ופונטים גרפיים) לשימוש בתכניות WISDOM. למסכים אלו ניתן לקרוא מתוך תכניות WISDOM. בנוסף, ניתן להפעיל את התוכנה כדי לקרוא את מסכי העזרה המוגדרים ל-WISDOM עצמו.
המערכת מורכבת משני חלקים:

א. תוכנה של עריכת ספריות מסכי עזרה.
בעזרת התוכנה, יכול המשתמש לבנות מסכי עזרה לשימוש בתכניותיו השונות, תחת WISDOM. מסכים אלו יכולים "לקרוא" זה לזה על פי הגדרות הבניה.
לתוכנה מצורפת ספרית מסכי עזרה, שנבנתה בעזרת התכנית עצמה.

ב. תוכנת תצוגת מסכי עזרה.
מסכי העזרה יכולים להקרא מתוך תכנית פרולוג בעזרת פרדיקט מיוחד. המשתמש מגדיר את שם הספריה הרצויה ובמידת הצורך גם את מספר המסך איתו תעלה התוכנה. לאחר כניסה למערכת העזרה, יכול המשתמש "לטייל" בין המסכים השונים.
ספריה מיוחדת של מסכי עזרה ל-WISDOM עצמו מסופקת בצמוד לתוכנה והיא ניתנת להפעלה גם מחוץ ל-WISDOM, כתוכנה עצמאית.



איור ב' – מסך עזרה ראשי של WISDOM

7. ניהול של שיגרות פסק (interrupts)

למשתמש של WISDOM-PC ניתנת גישה לפונקציות של BIOS ו-DOS תוך שינוי וקריאה של אוגרים (registers) שונים. נציג מספר דוגמאות:

א. BIOS interrupts

```
/*                                      הצב סמן על שורה L ועמודה C */
cursor(L,C) :-
    set_rx(2,0,0,0,0,0,L,C), int86(16).
/* גלגל מסך בכיוון Dir (מעלה/מטה), Lines שורות, החל משורה */
/*   Lt ועד שורה Lb. החל מעמודה Cl וכלה בעמודה Cr, מלא את */
/*                     המקום המתפנה ב-Attr Attribute */
scroll(Dir,Lines,Lt,Cl,Lb,Cr,Attr) :-
    set_rx(Dir,Lines,Attr,0,Lt,Cl,Lb,Cr), int86(16).
/*                                                  נקה מסך שלם */
cls :- scroll(7,0,0,0,24,79,7), cursor(0,0).
```

ב. DOS interrupts

```
/*                                              מצא תאריך נוכחי */
date(DayOfWeek,Year,Month,Day) :-
    set_rx(42,0,0,0,0,0,0,0), intdos,
    get_rx(DayOfWeek,_,_,_,Ch,Cl,Month,_), regX(Year,Ch,Cl).
/*                                  בנה אוגר X מבית H ובית L */
regX(Xval,Hval,Lval) :-
    bitleft(Hval,8,Hval1), bitor(Hval1,Lval,Xval).
```

פונקציות נוספות מתוארות בקבצים המלווים את WISDOM, לצורך בנית קבצים, בקרת מדפסת, תקשורת וכדומה.

נתאר שלד-תוכנה לבנית עורך (editor) בעזרת SWM. התוכנה ניתנת להרחבה כך שתמלא פונקציה של מעבד תמלילים מלא.

```
/* The SWM : A short demonstration.
   The following file is a simple edit-&-consult program, that allows
   editing of a file from within WP, while using its simplified WM functions.
   Remember, that as we allocate space for a new window, we need to load
   wpwm3 (the window-manager) with more space (the VS allocated here requires
   loading wpwm3 with -p35).

   The motivation is to have a tool for easying the edit&consult process -
   - we want to be able to load a file, test it, edit & change if needed,
   and update our program's knowledge as we do so.

   We shall define two predicates :
        1. edit(File) - to edit a file and then consult it, i.e., add its
           clauses to those in our data-base.
        2. reedit(File) - to edit a file and then REconsult it - we want
           the newly-edited clauses to replace those in our data-base.
```

```
 The different steps are:
       - set CO -> open the editor window
       - open file + read-in -> if the file to edit exists, we should read
                              it in.
       - make changes -> using the SWM editing facilities
       - write-out -> after editing, store the new file.
       - close file + close CO
       - at last, we can (re)consult the new file.
*/
edit(File) :-   % edit & consult
       edit_(consult(File)).

reedit(File) :- % edit & reconsult
       edit_(reconsult(File)).

edit_(G) :-
       abolish($ew, 1),              % erase possible previous clause.
       arg(1, G, File), atom(File),  % get file name
       setwedit(CW, W, File),        % build edit window
       assert($ew(W)),               % if we abort, we need to be able to
                                     % erase the editing window
       edit(File, W, Done),          % pass File-name & CO#, get Done arg.
       finwedit(W, CW),              % delete CO
       (var(Done) ; call(G)).        % if we exited correctly, (re)consult
edit_(_) :-    % reset mess..
       ewm_del, fail.

ewm_del :-      % we can call this predicate directly if we aborted !
       $ew(W), wmdelete(W).

setwedit(CW, W, Name) :-        % create editing CO
       wmcw(CW), wmcreate(2, 0, 80, 23, 80, 92, 4, Name, W), wmno(W).

finwedit(W, CW) :-              % delete CO, return to original.
       wmdelete(W), wmno(CW).

/* The process used here is relatively fast but leaves much to be improved.
   A much better (though slower) approach, would be to test every key-stroke
   (using getch/1 or get/1) and decide if it should be echoed to the window
   or processed otherwise. Here however, we chose to read and test lines.
   This requires the tested Ctrl-keys to be pressed at the begining of lines,
   at the end of the written part of the VS. (Ctrl-PgDn)
*/
edit(Name, W, Done) :-
       ldfile(Name),            % load file if it exists
       repeat,                  % until exit with edit-abort/exit.
       wmgets(255, Line),       % get maximum line length
       name(Line, List),
       efinish(List, Name, Done).    % check to see if special handling
                                     % is required.

ldfile(Name) :-         % read-in file
       exists(Name), !, wmreadin(Name).
ldfile(_).

/* We take care of 3 special requests :
*/
efinish([5|_], Name, 1) :-     % Ctrl-e to save & exit, so we set Done
       ewrite(Name).
efinish([2|_], Name, _) :-     % Ctrl-b to save & continue
       !, ewrite(Name), fail.
efinish([1|_], _, _).          % Ctrl-a to quit (abondon)

ewrite(Name) :-
       wmgcrs(L,_), wmscrs(L-1, 0),    % return 1 line to prevent writing
       wmwriteout(Name).               % the control character.
```

```
        One of WP's strongest features is its user-hooks. We shall define
        here a context-free grammar, and link it to the WP parser using the
        expand_clause/2 hook.

        The logical inference machine, the base of the Prolog language, has
        proved to be a powerful means for parsing and generating natural
        languages. (In fact, this was the origin of the Prolog language -
        an attempt to use logic to express grammar rules and formalize the
        parsing process !!)
        They can be built using a definite clause logic - we shall
        describe here such a definite clause grammar - a syntactic sugar to
        extend the ordinary Prolog syntax. DCGs are useful for both describing
        the language as for analyzing it.

        The following code was written for WP by Dr. L. Sterling.

        translate(grammarRule,PrologClause) :-
               PrologClause is the prolog equivalent of the
               context_free grammar rule grammarRule.

*/

:- op(60,xfy,'\').

/* A sentence will be accepted by the DCG parser, if it consists of a noun
   phrase followed by a verb phrase.
*/
$translate((Lhs --> Rhs),(Head :- Body)) :-
       $translate(Lhs,Head,Xs\Ys), $translate(Rhs,Body,Xs\Ys),!.

/* Add the difference-list to each nonterminal symbol
*/
$translate((A,B),(A1,B1),Xs\Ys) :-
       $translate(A,A1,Xs\Xs1), $translate(B,B1,Xs1\Ys),!.

/* extra tests for the righthand side of the grammar rule are not changed
   by the DCG translation process
*/
$translate(in_curly_brackets(G),G,S\S):- !.

$translate(Xs,true,S) :-
       terminals(Xs), !, sequence(Xs,S).

$translate(A,A1,S) :-
       compound(A), !, A =.. [F,Args], A1 =.. [F,S,Args].


non_terminal(A) :- atom(A).

terminals([X|Xs]).
terminals([]).

sequence([X|Xs],[X|S]\S0) :-
       sequence(Xs,S\S0).
sequence([],Xs\Xs).

/* make all rules visible to WP's parser - use HOOK !
*/
expand_clause((A --> B),(X :- Y)) :-
       $translate((A --> B),(X :- Y)).
```

```
<<<Input Suspended>>> [F1 - Drag] [F2 - Resize] [F9 - CopyActive] [F10 - Next]
            PATIENT                                  ANALYSIST
 [look,it,is,tracing,on,windows!  ser]

          Dialog Window                        Trace Window
   match(Response,Table,Ou  tracing: ¦ ?- eliza.
   wmno(0),                 0] call: eliza?
   reply(Output),           [1] call: eliza_won(I1768,O1776,CW1784)?
   wmno(I),                  [2] call: wmcreate(2,2,30,20,38,30,1,PATIENT,I27
   read_sentence(Input1),    [2] exit: wmcreate(2,2,30,20,38,30,1,PATIENT,3)?
   eliza(Input1,I,O).        [2] call: wmcreate(2,40,30,20,39,30,1,ANALYSIST,
reply([Head¦Tail]) :-        [2] exit: wmcreate(2,40,30,20,39,30,1,ANALYSIST,
   write(Head),              [2] call: wmcw(CW2768)?
   write(' '),               [2] exit: wmcw(0)?
   reply(Tail).              [2] call: wmno(3)?
reply('[]') :-               [2] exit: wmno(3)?
   nl.                      [1] exit: eliza_won(3,4,0)?
                            [2] call: read_sentence(Input1808)?
yes                          [3] call: read(Xs9720)?
¦ ?- trace.                  [3] exit: read([look,it,is,tracing,on,windows])?
```

איור ג' – ניפוי של ELIZA בעזרת החלונות

```
              PATIENT                                ANALYSIST
¦ [i,feel,tired].                       do you often feel that way ?
¦ [since,i,met,my,brother].             can you tell me more about brother
¦ [no].                                 please go on
¦ [i,like,your,style].                  does anyone else in your family like my
                                         style ?
¦ [yes].                                please go on
¦ [i,think,wisdom,prolog,is,great].     what makes you think so ?
¦ [you,teeze,me].                       what makes you think i teeze you ?
¦ [i,hate,you].                         please go on
¦ [bye].                                Goodbye. I hope I have helped you
```

איור ד' – שימוש בחלונות: דו-שיח על בסיס תכנית ELIZA

8. הפעלת התכנית

WISDOM [state-file] -s⟨stack-size⟩ -d⟨data-base⟩ -w

הסברים:

- state-file – שם קובץ המידע הראשוני, אותו תעלה המערכת (ברירת המחדל היא קובץ stdsts).
- ⟨state-size⟩ – מספר יחידות הזיכרון שתוקצנה למחסנית.
- ⟨data-base⟩ – מספר יחידות הזיכרון שתוקצנה למאגר הנתונים.
- w – תחת PC בלבד, לעבודה עם חלונות.

Note, that using any predefined predicate (evaluable predicate) whose name starts with a $ sign, requires the use of an unlocked state, even when working as shown in (1).

1. Suppose you want to use the 'b' option in the tracer, to load a temporary shell. It might look like this :

```
bshell :- repeat, % if we used abort, the tracer state will be lost !
    (ttynl, display('Next? '),   % make sure you remeber it's 2nd shell
    sread(Goalb, Vs), s_expand_goal(Goalb,Goal), !),
    process(Goal,Vs).
process(exit,_).    % return to tracer
process(Goal,Vs) :-
    user_call(Goal,Vs), !, fail.
```

2. Programs that use recursion for their main goal, might eventually have stack & DB overflow problems - WP does the actual freeing of buffers just before returning to the shell.
Consider for example the 'Framework for playing games' - Program 18.8 in 'The Art of Prolog' :

```
play(Game) :-
    initialize(Game, Position, Player),
    display_game(Position, Player),
    play(Position, Player, Result).

play(Position, Player, Result) :-
    game_over(Position, Player, Result), !, announce(Result).
play(Position, Player, Result) :-
    choose_move(Position, Player, Move),
    move(Move, Position, Position1),
    display_game(Position1, Player),
    next_player(Player, Player1),
    !, play(Position1, Player1, Result).
```

After a few moves, the stack ('vars' buffer) will overflow !

The following code will correct the situation:

```
play(Game) :-
    initialize(Game, Position, Player),
    display_game(Position, Player),
    abolish($position,2),              % erase previous positions
    assert($position(Position, Player)),   % store init. position
    assert((shell :- play)).           % change WP's shell !!

play :-
    retract($position(Position, Player)),   % retrieve position
    game_over(Position, Player, Result),
    !,
    abolish(shell, 0),       % game is over -> return to default shell
    announce(Result) ;
    play(Position, Player, Result). % game not finished -> continue

play(Position, Player, Result) :-
    choose_move(Position, Player, Move),
    move(Move, Position, Position1),
    display_game(Position1, Player),
    next_player(Player, Player1),
    !,
    assert($position(Position1, Player1)),  % store new position
    abort.  % free stack & abolished data, return through modified shell.
```

Note, that exiting here with an abort or Ctrl-C without previously re-setting the default shell, will cause continuous try of the game. You should therefor allow for a user-stop at the game_over/3 level. (modify the predicate itself to ask for a user-break or add an ORed clause in the play/0 predicate)

נספח ב'

אופרטורים

את פעולות החשבון של פרק 8 ואת הפרדיקט not של פרק 11 הצגנו בצורה של אופרטורים. ניתן להגדיר בצורה זו כל פרדיקט של אריטי 1 או 2. כדי להגדיר אופרטור יש לציין את שלושת המאפיינים: קדימות, סוג של התחברות ושם.

א. קדימות (precedence)

קדימות של אופרטור מוצגת באמצעות מספר שלם. בגירסת WISDOM-PROLOG המספר הזה נע בין 1 לבין 1200. נציין כי בגירסה זו הקדימות של האופרטור "+" הינה 500, והקדימות של האופרטור "*" הינה 400. באופן כללי, האופרטור בעל הקדימות הגבוהה ביותר מתקבל בתור הפנקטור הראשי, אשר על כן הביטוי 2+3*5 מתפרש על ידי מערכת הפרולוג כביטוי מורכב +(2,*(3,5)).

ב. סוג של התחברות (association)

הסוגים של האופרטורים: infix, prefix, postfix נידונו בפרק 8. בין שני אופרטורים בעלי קדימות זהה מבדילים על ידי קביעת סוג של התחברות. את סוג ההתחברות קובעים בעזרת הסימונים האלה: את האופרטור מסמנים ב-f, את הארגומנט בעל קדימות הנמוכה מזו של f, מסמנים ב-x, ואת הארגומנט אשר הקדימות שלו אינה נמוכה מזו של f מסמנים ב-y. בעזרת סימונים אלו ההתחברות של האופרטורים מסומנת כך:

אופרטורים מסוג infix:

xfx – הקדימות של הארגומנטים נמוכה מזו של האופרטור.
xfy – הקדימות של הארגומנט השמאלי נמוכה מזו של האופרטור.
yfx – הקדימות של הארגומנט הימני נמוכה מזו של האופרטור.

אופרטורים מסוג prefix:

fx – הקדימות של הארגומנט נמוכה מזו של האופרטור.
fy – הקדימות של הארגומנט אינה נמוכה מזו של האופרטור.

לסימון של האופרטורים מסוג postfix: xf, yf, ניתן פירוש דומה.

נציג שתי דוגמאות:

1. אם ההתחברות של "-" הינה yfx, אז הביטוי 7-4-1 מתפרש כ-1-(7-4).

2. אם ההתחברות של not הינה fy, אז הביטוי not not a מתפרש כ-not(not a).
אם ההתחברות של not היא fx, אז הביטוי not not a אינו חוקי: הביטוי הנכון במקרה זו הוא not(not a).

ג. שם (name)

השם של האופרטור הינו אטום.

את האופרטור op מגדירים באופן הבא:

```
|?- op(200, xfx, likes).
|?- op(100, xfy, and).
```

באמצעות האופרטורים שהגדרנו ניתן לרשום את הביטוי:

likes(tom, and(mary, ann))

באופן הבא:

tom likes mary and ann

כמו כן ניתן לרשום את הביטוי:

likes(tom, and(mary, and(ann, liz)))

באופן כדלהלן:

tom likes mary and ann and liz

נציין כי כמה אופרטורים בעלי קדימות והתחברות זהות ניתן להגדיר באמצעות רשימה.

להלן אוסף של האופרטורים שהוגדרו מראש במערכת WISDOM-PROLOG. הסימון ":-" לפני ההגדרה משמש לציון של שאילתה (ראה איור 10.1 של פרק 10).

```
:-op(1200, fx, [:-, ?-]).
:-op(1200, xfx, [(:-), <-, -->]).
:-op(1100, xfy, ';').
:-op(1000, xfy, ',').
:-op(900, fx, [not]).
:-op(700,xfx,[=,\=,is,:=,=..,==,\==,=:=,=\=,<,>,=<,>=]).
:-op(500, yfx, [+, -]).
:-op(500, fx, [(+), (-)]).
:-op(400, yfx, [*, /, //]).
:-op(300, xfx, [mod]).
```

נסכם ונאמר: במסגרת נספח זה הדגמנו רישום של אופרטורים בתעתיק של פרולוג. אלה מן הקוראים אשר מעוניינים להרחיב את הדיון בנושא אופרטורים בשפות תכנות, יוכלו למצוא חומר לימודי רב בנושא, למשל סעיף 4.6 בספר [Aho & others, 1986], או פרקים 11, 12 בספר [Gries, 1971]

```
WISDOM OPERATORS                                                    [1]
An OPERATOR is defined in the form:
op(precedence, type, name_list) where

precedence is an integer from 1 to 1200 defining the operators priority -
         the HIGHER the precedence the LOWER the priority of evaluation.

type is either PREFIX (fx, fy), INFIX (xfx, yfx, xfy) or POSTFIX (xf, yf)
     the x and y define OPERANDS (arguments) and the f the OPERATOR
     fx - means that the precedence of the OPERAND must be LOWER than that
          of the operator.
     fy - means that the precedence of the OPERAND can be EQUAL to that
          of the operator.
     xfx -> a-b-c -> illegal  -  neither a-b nor b-c have lower precedence
                                 relative to each other.
     yfx -> a-b-c -> (a-b)-c  -  -(-(a,b),c)
     xfy -> a-b-c -> a-(b-c)  -  -(a,-(b,c))

name_list is an ATOM or a list of atoms defining the name by which the
         operator will be called.
                                       System Operators        Prev Scr
```

איור ה' – מסך עזרה של WISDOM: הגדרת אופרטורים

```
SYSTEM OPERATORS                                        [101]
precedence     type     name

1200             fx     :-  ?
1200            xfx     (:-)  <-   -->
1100            xfy     ;
1000            xfy     ,
 950             fx     if
 945            xfy     then
 940            xfy     else
 900             fx     not
 700            xfx     =  \=  is  :=  =..  ==  \==  =:=  =\=  <  >  =<  >=
 500            yfx     +  -
 500             fx     (+)  (-)
 400            yfx     *  /  //  div
 300            xfx     mod
                      Prev Scr
```

איור ו' – האופרטורים של WISDOM

נספח ג'

סביבת עבודה TI - PROLOG

נמסר באדיבות: טקסל אלקטרוניקה בע"מ
דרך פתח-תקוה 37, תל-אביב

1. תאור המוצר וסביבת הפיתוח

חברת TEXAS INSTRUMENTS מציגה את TI-PROLOG, המשולב בסביבת העבודה של ה-EXPLORER (מכונת LISP של TEXAS INSTRUMENTS). TI-PROLOG הוא קבוצת-על של DECsystem-10 PROLOG ועל כן הוא תואם את רוב יישומי הפרולוג הקיימים (כלומר הוא פרולוג תיקני; אין סטנדרט דה-יורה אבל יש דה-פקטו, אשר מזוהה עם DECsystem-10 ומכונה בשם "אדינבורו").
אותה מידת תמיכה וסיוע הניתנים למתכנת LISP, ניתנת גם למתכנת ה-PROLOG ב-EXPLORER.

ה-PROLOG של TI מספק:

- עורך לקלט – הרחבה של ה-ZMACS EDITOR לתמיכה בשפת ה-PROLOG.
- השלמת שמות קבצים – PATHNAME.
- השלמת שמות פרדיקטים של המערכת ושל המשתמש.
- הסברים לכל הפרדיקטים של המערכת, עם אפשרות נוחה להוסיף הסברים גם לפרדיקטים שמוגדרים על ידי המשתמש.
- מישק עבור UNIVERSAL COMMAND LOOP כמו במערכות המשנה האחרות ב-EXPLORER.
- אפשרות לשמור ולדפדף בהסטוריה של הידברות (interaction) עם המערכת.

– PROLOG COMPILER, שמופעל אוטומטית על כל הפיסקאות המוקלדות.

2. תכונות נוספות

– התאמות במיקרוקוד של ה-EXPLORER להשגת ביצועים ברמה גבוהה.

– קומפילציה אינקרמנטלית: ניתן להדר (compile) חלקים נבחרים מבלי להדר את התכנית כולה.

– עזרים בנוסח DECsystem-10, הכוללים DEFINITION. TRACING, SPYPOINTS, LOOK UP

– טיפול בשגיאות נעשה באמצעות מערכת ה-LISP הסטנדרטית.

– אפשרות בלתי מוגבלת ל-ASSERT, CLAUSE ו-RETRACT לקוד שעבר הידור.

– אין צורך ב-DECLARATIONS עבור פרוצדורות דינמיות, מאחר וניתן לראות את כולן כדינמיות.

– אופטימיזציה של TAIL RECURSION.

– ניהול הזכרון נעשה באופן אוטומטי.

– על-מנת לספק שילוב מוחלט בין LISP ו-PROLOG, ניתן לקרוא ל-LISP מ-PROLOG ול-PROLOG מ-LISP וזאת משום שלשניהם ייצוג נתונים משותף. מאחר והגישה ל-LISP כה נוחה, TI-PROLOG מאפשר גישה מלאה למערכות התוכנה הנוספות של ה-EXPLORER, כמו למשל ל-WINDOW SYSTEM.

– TI-PROLOG מספק כמה תכניות לדוגמה המתארות את עוצמת המערכת. למשל, REST (Rule-based Expert System Tool) הינה קונכיה של מערכת מומחה הכתובה ב-PROLOG.

– על-מנת לאפשר כתיבה של תכניות PROLOG גדולות, ניתן לארגן מודולים שעברו הידור ולטעון אותם באמצעות מנגנון, המאפשר טעינת מודולים של LISP.

TI-PROLOG בנוי מתוכנת מיקרוקוד ותוכנת LISP, ואין כל צורך בשינויים או תוספות לחומרה.

הדרישות: מערכת התוכנה EXPLORER במהדורה 2.1 ואילך.

```
/*****************************************************************************/
/* The following is a demonstration in the Prolog to Lisp                   */
/* interface. Use the consult command on this file and                      */
/* the classical problem "Towers of Hanoi" will run.                        */
/* The graphics will be displayed in the graphics window on the top of the screen and */
/* a printed output will be displayed in the Lisp listener below it.        */
/*                                                                          */
/* Read the read-me.text file to know how to run the demo and to get more information */
/*                                                                          */
/*****************************************************************************/


:- defpackage(hanoi).

:- eval(['LOAD', 'TI-Prolog:Starter-Kit;Hanoi'],_).

hanoi(N) :-
   apply('HANOI':'SET-SCREEN',[N],Output_window),    % if N is too big then Output_window
   Output_window \== [],                              %    is nil and this predicate fails
   repeat,
   eval(['HANOI':'NEW-GAME'],_),
   \+ move_disks(N, left, center, right,Output_window),  % repeat until END key pressed
   eval(['HANOI':'RESET-SCREEN'],_).

move_disks(0,_,_,_,_) :- !.

move_disks(N, Pole1, Pole2, Pole3, Output_window) :-

   write('Prolog : move_disks (',Output_window),       %trace Prolog
   write(N    ,Output_window),
   write(' , ',Output_window),
   write(Pole1,Output_window),
   write(' , ',Output_window),
   write(Pole2,Output_window),
   write(' , ',Output_window),
   write(Pole3,Output_window),
   write(')',Output_window),
   nl(1,Output_window),

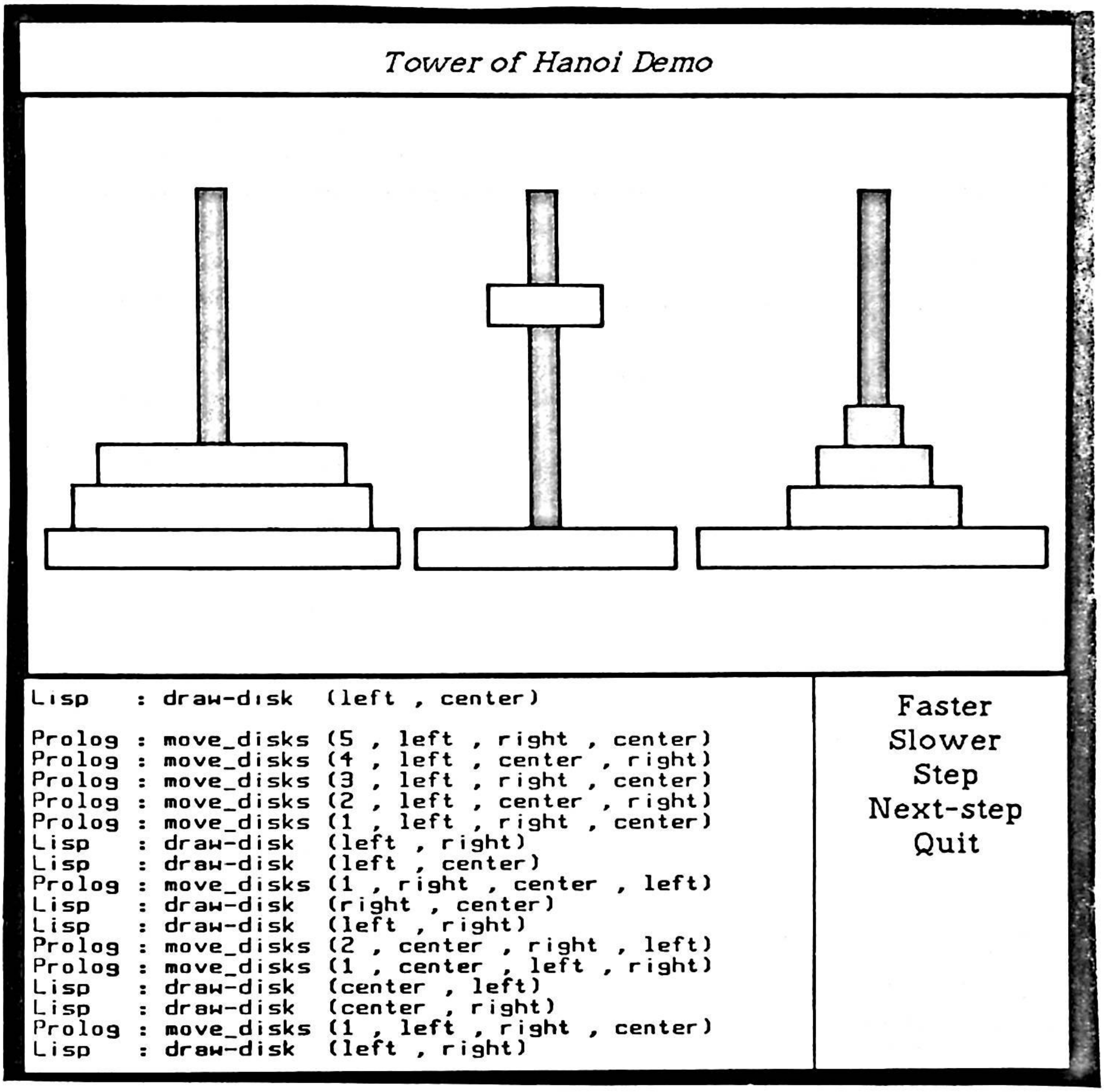
   M is N-1,

   move_disks(M, Pole1, Pole3, Pole2, Output_window),

   write('Lisp   : draw-disk  (',Output_window),        %trace Lisp
   write(Pole1,Output_window),
   write(' , ',Output_window),
   write(Pole2,Output_window),
   write(')',Output_window),
   nl(1,Output_window),

   !,
   eval(['HANOI':'DRAW-DISK',
         ['QUOTE',Pole1],['QUOTE',Pole2]],[]),          % fails if end key pressed
   move_disks(M, Pole3, Pole2, Pole1, Output_window).
```

תכנית ו' – מגדלי האנוי ב-TI-PROLOG



איור ז' – סביבת עבודה של TI-PROLOG

נספח ד'

הפרוצדורה append בפסקל ובפרולוג

בפרק 9 הגדרנו את המושגים הבאים של פרולוג: פרוצדורה (procedure), רשימה (list). קוראים רבים יודעים, שהמושגים האלה מקובלים גם בשפות פרוצדורליות. בנספח זה נשווה בין append בשפה פרוצדורלית לבין append של פרולוג.

הפרוצדורה append בשפה דמויית PASCAL:

```
procedure append(L1, L2: LIST; var L3: LIST);
        begin local T: LIST
          if L1 = nil then L3 := L2
          else
            begin
              append(tail(L1), L2, T);
              L3 := cons(head(L1), T)
            end
        end;
```

הסבר:
הפרמטרים L1, L2 – מצביעים (pointers) לרשימות (lists).
הרשימה – רשומה (record), מכילה שני שדות (fields).

את השדות של הרשימה L נסמן ב-head(L) ו-tail(L). השדה head מצביע על רשימה או על אטום **כלשהו**, והשדה tail מצביע על רשימה או על אטום **מסוים**, הקרוי nil (בפרק 9 סימנו אטום זה ב-[]). הפונקציה cons(head,tail), דמויית LISP, בונה את הרשימה (בפרק 9 סימנו פונקציה זו ב-dot).

הפרמטר L3 – מכיל את תוצאת השרשור של L1 ו-L2.
המשתנה X – מכיל את תוצאת הביניים.

נעבור עתה לפרוצדורה append של פרולוג. ההשמה (assignment) וההסתעפות המותנית (conditional branching) של פסקל הוחלפו במנגנון של פרולוג, המכיל חיפוש, התאמה/איחוד ונסיגה. כתוצאה מכך, נגרמו השינויים הבאים:

1. הפרוצדורה של פסקל קלטה L1 ו-L2 והחזירה פלט L3. לעומת זאת, בפרוצדורה של פרולוג הפרמטרים L1, L2, L3 איבדו את המשמעות של קלט-פלט, שכן ניתן להציג גם שאילתה מסוג זה:

```
?-  append(L1, L2, [a,b,c]).
L1 = []
L2 = [a,b,c]
```

2. כמו כן, הפרוצדורה של פרולוג מחזירה תשובה, שמכילה משתנים:

```
?-  append([a,b], L2, L3).
L2 = L280
L3 = [a, b| L280]
```

3. ובנוסף, הפרוצדורה של פרולוג מסוגלת לייצר יותר מתשובה אחת:

```
?-  append(L1, [a,b], L3).
L1 = []
L3 = [a,b];

L1 = [L180]
L3 = [L180, a, b]
```

לקורא, אשר מעוניין להמשיך בהשוואה בין שפה פרוצדורלית כדוגמת פסקל, לבין פרולוג, נמליץ על המאמרים האלה: [Cohen, 1985], [Williams & Chen, 1986].

נספח ה'

סביבת עבודה TURBO-PROLOG

נמסר באדיבות: גושן מחשבים ומערכות
ת.ד. 33
קרית אונו 55100

1. טורבו-פרולוג (גירסה 1.0)

טורבו-פרולוג הינה שפת מחשב של הדור החמישי המביאה לתכנות מימד חדש. בגלל גישתה הטבעית וההגיונית, הן אנשים חדשים והן מתכנתים מקצועיים, יכולים לבנות יישומים בעלי עוצמה – כגון מערכות מומחה, בסיסי ידע ע"פ הזמנה, מישקים לשפה טבעית ומערכות ניהול מידע מתוחכמות.
טורבו-פרולוג מתאים לפרולוג הסטנדרטי, הקרוי "אדינברו".

חבילת התוכנה של טורבו-פרולוג כוללת ספר הדרכה בן 200 עמודים, תכנית עזר המסבירה צעד אחר צעד כיצד להשתמש בטורבו-פרולוג. כמו כן, מסופקת מערכת-עזר "HELP" מלאה עבור העורך. את כל הפעולות ניתן לבצע דרך תפריטי "PULL DOWN".

כמו טורבו-פסקל של חברת בורלנד שנמצאת בשימוש רחב בכל העולם, גם טורבו-פרולוג מספקת סביבת תכנות מוכללת מלאה. עורך תמלילים הנמצא בחלון, עובד יחד עם המהדר כדי ליצור קוד מקור מושלם.

* **מהדר הגדלה מתקדם.**
המהדיר מעל ל-2500 שורות בדקה על IBM PC הסטנדרטי, ויוצר קוד (מכונה) מקומי ומודולי-יעד הניתנים לחיבור (בתהליך LINK). פורמט החיבור תואם ל-LINKER של ה-PC-DOS. התכניות מהודרות לקבצי EXE ויכולות להיות בגודל של עד 640K. ניתנת תמיכה לקבצי INCLUDE.

* **עורך תמלילים הידברותי**
מאפשר התרת "באגים" בקלות ובמהירות. כאשר המשתמש מהדר את תכניתו, העורך ממקד את הסמן אוטומטית בתכנית המקור, שבה נמצאת הטעות.

* **סביבה רבת-חלונות מיוחדת**
מאפשרת לנוע קדימה ואחורה בין החלונות. היא מציגה את העורך, תהליך המעקב, תכנית המקור, מטרות והתוצאות של פרולוג – הכל בו-זמנית. המשתמש יכול לשנות את הגודל והצבע של התכניות, כדי שיתאימו לסביבת התכנות שלו.

* **גישה ותמיכה מלאה של DOS.**
המשתמש יכול לתת כל פקודת DOS בתוך תכנית טורבו-פרולוג שלו. הוא אפילו יכול להריץ תוכנה אחרת מבלי לעזוב את תכניתו.

* **הכללת עורך טורבו-פרולוג בתכנית.**
בזמן ריצה, תכניות טורבו-פרולוג יכולות לקרוא לעורך, המאפשר לצפות בקבצי תמליל ולערוך אותם.

* **תמיכה מלאה בחלונות גרפיקה וטקסט.**
מאפשרת ליצור סביבה רבת-חלונות עבור תכניות המשתמש. ניתן להזיז ולשנות גודל וצבע בכל החלונות.

* **ניידות**
טורבו-פרולוג פועל במחשבי IBM PC/XT/AT תואמי PC ו-IBM-PS/2 תחת MS-DOS או PC-DOS (גירסה 2.0 ומעלה). ניתן לגשת בנקל לזיכרון ה-PC ולחיבורי ה-I/O. ניתן לחבר מודולים בקוד מכונה לתכניות טורבו-פרולוג של המשתמש.

* **בסיס נתונים גיאוגרפי** GEOBASE לשפת שאילתות טבעית
המכיל מידע גיאוגרפי של USA- נהרות, ערים, הרים ודרכים. מושלם ומוכן להידור, מכיל תכניות מקור מתועדות. ניתן להשתמש בבסיס נתונים זה כמו שהוא, או לשנות אותו לפי הצורך. בסיס נתונים זה מאפשר ללמוד על טורבו-פרולוג.

2. טורבו-פרולוג טולבוקס

"טורבו-פרולוג טולבוקס" הינו מוצר נוסף של חברת בורלנד הכולל שישה מרכיבים בחבילת תוכנה אחת המכילה כלים עבור:

* עיצוב תפריטים,
* הכנת מסכים,
* גרפיקה לבתי עסק,
* העברת קבצים מהתכנות הבאות: REFLEX, dBASE-III PLUS, לוטוס 1-2-3 ו-SYMFONY,
* תקשורת,
* ניתוח דיקדוקי.

טורבו-פרולוג טולבוקס מוסיף כח רב לטורבו-פרולוג ע"י יותר מ-80 כלים (8000 שורות של קוד מקור). כדי לסייע למשתמש המתחיל, חבילת התוכנה כוללת 40 דוגמאות של תכניות, אשר מדגימות כיצד להשתמש בכלים אלו של בינה מלאכותית.

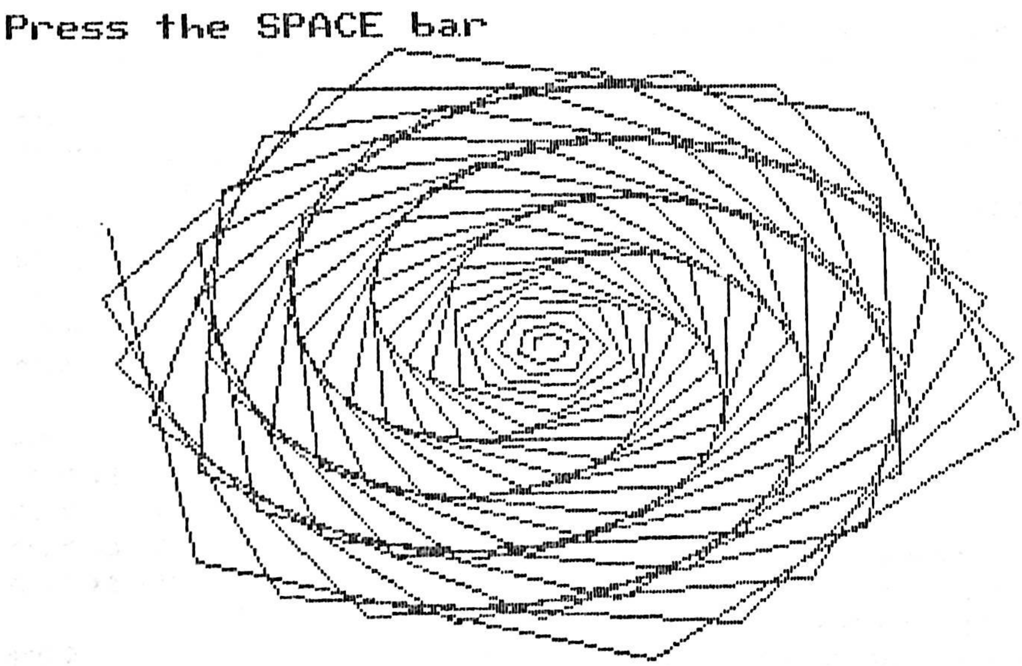
```
   Run      Compile     Edit      Options      Files      Setup      Quit

 ─────────────────── Editor ─────────────────
  Text:  166  Free:65209  Indent  Insert  MYFIT.PRO   ── Dialog ──
predicates
  polyspiral(integer)
goal
  graphics(1,1,4),polyspiral(400).
clauses
  polyspiral(X) :- forward(X),right(62),
      Y=X+100,Y<15000,polyspiral(Y).

 ──────── Message ────────              ──────── Trace ────────

Use first letter of option  or  select with  ->  or  <-
```

איור ח׳ – המסך של טורבו-פרולוג בעת עבודה עם תכנית



איור י׳ – הרצה של תכנית בטורבו-פרולוג

רשימה של תכניות ופרוצדורות

תכנית 4(ט) – בונה-מבזבז-מתרושש
תכנית 5(א) – חישוב העצרת
תכנית 5(ב) – חישוב עצרת בשיטת צבירה איטרטיבית
תכנית 5(ג) – עצרת איטרטיבית עם הלולאה
תכנית 6(א) – מפת היעדים (גירסה עברית)
תכנית 6(ב) – מפת היעדים (גירסה אנגלית)
תכנית 7(א) – החיפוש לעומק
תכנית 7(ב) – החיפוש לעומק, כאשר המצב הסופי מוגדר מראש
תכנית 8 – חיפוש דרך במבוך
תכנית 9(א) – חיפוש אחר שביל
תכנית 9(ב) – חיפוש שביל ללא לולאות
תכנית 9(ג) – חיפוש עם מעבר מתוחכם
תכנית 9(ד) – חיפוש עד עומק מסוים
תכנית 10(א) – פתרון בעית שמונה המלכות
תכנית 10(ב) – פתרון בעית המלכות באמצעות "ייצר-ובדוק"
תכנית 10(ג) – פתרון בעית המלכות באמצעות select
תכנית 11(א) – מגדלי האנוי
תכנית 11(ב) – מגדלי האנוי (גירסה שניה)
תכנית 12 – התפתחות של בעלי-חיים, מודולים א', ב', ג'
תכנית 13 – מערכת מומחה לזיהוי חיות:
א. מישק המשתמש
ב. מסד ידע
תכנית 14 – קונכיה של מערכת מומחה, מודולים א'-ו'

נספח א'

תכנית א' – טיפול במערכים תחת WISDOM
תכנית ב' – שתי גירסאות של ELIZA: עם ובלי חלונות
תכנית ג' – editor
תכנית ד' – דוגמה של DCG
תכנית ה' – שינויים ברמת ה-shell

נספח ג'

תכנית ו' – מגדלי האנוי ב-TI-PROLOG

ביבליוגרפיה

Aho, A.V. & Sethi, R. & Ullman, J. (1986): "Compilers - Principles, Techniques and Tools", Addison-Wesley Co.

Barnes, J. (1987): "Giving Prototyping Free Rein and Just Due", Computerworld, September 7, 21(36).

Barr, A. & Feigenbaum E.A. (1982): "The Handbook of Artificial Intelligence", Vol. 1, 2, 3, William Kaufmann Inc., Los Altos, CA.

Brooks, F. P. Jr. (1987): "No Silver Bullet - Essence and Accidents of Software Engineering", Computer, April, pp. 10-19.

Church, A. (1956): "Introduction to Mathematical Logic", Princeton University Press.

Cohen, J. (1985): "Describing Prolog by its Interpretation and Compilation", Communications of the ACM, 28, pp. 1311-1324.

Colmerauer, A. (1985): "Prolog in 10 Figures", Communications of ACM, 28, pp. 1296-1310.

Filgueiras, M. (1984): "A Prolog Interpreter Working with Infinite Terms", Implementations of Prolog, Ellis Horwood Limited, Chichester, England.

Fuchi, K. (1984): "Revisiting Original Philosophy of Fifth Generation Computers Systems Project", Proc. Inter. Conf. of FGCS, ICOT, Tokyo.

Golumbic, M.C. & Markovich, M. & Tsur, S. & Shild, U. (1986): "A Knowledge-Based Expert System for Student Advising", IEEE Trans. on Education 29, pp. 120-123.

Gries, D. (1971): "Compiler Construction for Digital Computers", John Wiley & Sons, Inc.

Hayes-Roth, F. & Waterman, D.A. & Lenat, D.B. (1983): "Building Expert Systems", Addison-Wesley Co. Inc.

Hughes, G.E. & Cresswell, M.J. (1968): "An Introduction to Modal Logics", London: Methcuen.

Ishizuka, M. & Kanai, N. (1985): "Prolog-ELF Incorporating Fuzzy Logic", New Generation Computing, 3, pp. 479-486.

Jaffar, J. & Lasser, J.L. & Lloyd, J. (1983): "Completeness of the Negation as Failure Rule", Proc. of IJCAI, pp. 514-519.

Kawanobe, K. (1984): "Current Status and Future Plans of the Fifth Generation Computer Systems Project", Proc. Inter. Conf. of FGCS, ICOT, Tokyo, pp. 3-17.
Luconi, F.L. & Malone, T.W. & Scott Morton, M.S (1986): "Expert Systems: The Next Challenge for Managers", Sloan Management Review, Summer Fall, pp. 3-13.
Mamdani, E.H. & Efstathiou, H.J. (1985): "Higher-order Logics for Handling Uncertainty in Expert Systems", Int. J. Man-Machine Studies 22, pp. 283-293.
Manna, Z. & Waldinger, R. (1985): "The Logical Basis for Computer Programming", Vol 1: Deductive Reasoning, Addison-Wesley Co., Inc.
Martin, J. (1985): "System Design from Provably Correct Constructs", Prentice-Hall, New Jersey, pp. 17-34.
Nilsson, N.J. (1980): "Principles of Artificial Intelligence", Tioga Co., Palo-Alto, CA.
Papert, S. (1980): "Mindstorms", Basic Books, Inc., New-York.
Rich, E. (1986): "Artificial Intelligence", McGraw-Hill Inc.
Robinson, J.A. (1965): "A Machine Oriented Logic based on the Resolution Principle", J. of ACM 12, pp. 23-41.
Rossi, G. (1986): "Uses of Prolog in Implementation of Expert Systems", New Generation Computing 4, pp. 321-329.
Schrage, G. (1986): "The Eight Queens Problem as a Strategy Game", Int. J. Math. Educ. Sci. Technology 17, pp. 143-148.
Shapiro, E.Y. (1983): "A Subset of Concurrent Prolog and its Interpreter", Tech. Report CS83-06, Weizmann Inst. of Sci., Israel.
Shapiro, E.Y. (1984): "Alternation and the Computational Complexity of Logic Programs", Tech. Report CS84-06, Weizmann Inst. of Sci., Israel.
Shapiro, E.Y. (1986): "On Evaluating the Adequacy of a Language for an Architecture", Tech. Report CS86-01, Weizmann Inst. of Sci., Israel.
Sheil, B. (1987): "Thinking about Artificial Intelligence", Harvard Business Review 4, pp. 91-97.
Sterling, L. (1984): "Expert System = Knowledge + Meta- Interpreter", Tech. Report CS84-17, Weizmann Inst. of Sci., Israel.
Sterling, L. & Shapiro, E. (1986): "The Art of PROLOG", MIT Press, Cambridge, Mass.
Warren, D.S. (1984): "Database Updates in Pure Prolog", Proc. Inter. Conf. of FGCS, ICOT, Tokyo, pp. 244-253.
Williams, M.H. & Chen, G. (1986): "Translating Pascal for Execution on a Prolog-based System", The Computer J., 29, pp. 246-252.
Winston, P.H. (1984): "Artificial Intelligence", Addison-Wesley Co. Inc.
Wos, L. & Overbeek, R. & Lusk, E. & Boyle, J. (1984): "Automated Reasoning", Prentice-Hall Inc., New Jersey.
Yazdani, M. (1984): "New Horizons in Educational Computing", Ellis Horwood Limited, Chichester, England.
Zadeh, L.A. (1973): "Outline of a New Approach to the Analysis of Complex Systems and Decision Processes", IEEE Trans. Systems, Man and Cybernetics, SMC-3, pp. 28-44.

מילון מונחים עברי-אנגלי

conjuction AND	$\wedge$	גימום "גם"
disjunction OR	$\vee$	איווי "או"
negation NOT	$\neg$	שלילה "לא"
universal quantifier	$\forall$	כמת כולל (כללי)
existential quantifier	$\exists$	כמת יישי (קיומי)
union	$\cup$	איחוד
implication IF-THEN	$\rightarrow$	אימוז "אם-אז"
implication IF-THEN	$\leftarrow$	אימוז "אם-אז"
equivalence IF-AND-ONLY-IF	$\leftrightarrow$	אימום "אם-ורק-אם"
equivalence	$\Leftrightarrow$	שקילות
follows (implies)	$\Rightarrow$	נובע (גורר)
follows (implies)	$\Leftarrow$	נובע (גורר)

most general unifier (mgu)	210	(ה)מאחד הכללי ביותר
derived goal	117, 212	(ה)מטרה הנגזרת
prototype	19, 362	אב-טיפוס
ancestor	134, 147, 223	אב-קדמון
register	372	אוגר
operator	161, 379	אופרטור
atom	101, 143	אטום
disjunction OR	44, 113	איווי "או"
unification	207	איחוד (תהליך ה...)
implication IF-THEN	44, 101	אימוז "אם-אז"
equivalence IF-AND-ONLY-IF	44, 101	אימום "אם-ורק-אם"
infinite (tree, structure)	148, 288	אינסופי (עץ, מבנה)
requirement	19	איפיון
case study	155	אירוע
true	42, 59, 198	אמיתי
truth	42	אמת
search stategy	16, 298, 315	אסטרטגיה של חיפוש
axiom	75, 118	אקסיומה (הנחה)
argument	84, 144	ארגומנט (מתואר)
arity	145	אריטי (רבב)
occurs check	207	בדיקת נוכחות
term	101, 143	ביטוי
ground term	199	ביטוי בסיסי
artificial intelligence	15, 28	בינה מלאכותית
database	22, 32, 82, 85	בסיס (מסד) נתונים
knowledge base	32, 82, 337	בסיס (מסד) ידע
conjuction AND	44, 93	גימום "גם"
epistemological approach	22	גישה אפיסטמולוגית
declarative approach	22	גישה הצהרתית

מילון מונחים אנגלי-עברי

conjuction AND	$\wedge$	גימום "גם"
disjunction OR	$\vee$	איווי "או"
negation NOT	$\neg$	שלילה "לא"
universal quantifier	$\forall$	כמת כולל (כללי)
existential quantifier	$\exists$	כמת יישי (קיומי)
union	$\cup$	איחוד
implication IF-THEN	$\rightarrow$	אימוז "אם-אז"
implication IF-THEN	$\leftarrow$	אימוז "אם-אז"
equivalence IF-AND-ONLY-IF	$\leftrightarrow$	אימום "אם-ורק-אם"
equivalence	$\Leftrightarrow$	שקילות
follows (implies)	$\Rightarrow$	נובע (גורר)
follows (implies)	$\Leftarrow$	נובע (גורר)

סביבת עבודה VM/PROLOG

גירסת VM/PROLOG של IBM פועלת תחת VM/SP ומיושמת במחשבים ממשפחת 370 (30XX, 43XX, 9370).

נציין כמה מהאיפיונים החשובים של VM/PROLOG:

- המפרש של VM/PROLOG מכיל הרחבה (nucleus extension) של CMS. לפיכך, המשתמש בו נהנה מהשירותים של CMS, אשר מאפשר לו להפסיק את הפעילות בפרולוג, לעבור ל-CMS ולחזור להמשך העבודה.
- בעת הטעינה של מפרש פרולוג ניתן להגדיר את הגודל של המחסניות local, global, rule ו-trail, לטעון LISP/VM ולהגדיר אופציות נוספות.
- המשתמש יכול להגדיר פרדיקטים בשפת הסף ולהטעינם במידת הצורך.
- ניתן להאיץ את התכנית על-ידי הכרזת חוקים סטטיים. אלה הם החוקים, אשר אין צורך לשנותם בזמן ריצת התכנית.
- התוכנה מכילה אמצעי עקיבה וניפוי: TRACE, SPY, SPY_DISPLAY, BREAK ועוד. ניתן לשנות פרדיקטים אלה ולבנות שגרות ניפוי נוספות על-פי דרישות המשתמש.
- התוכנה מייעלת את השימוש בזיכרון באמצעות Garbage Collector ו-Tail Recunsion Optimization.
- תמיכה בנקודה צפה ובמספרים שלמים 32-bit.
- הגנה מפני התנגשות של שמות הקבצים בעת טעינתם.
- התוכנה מכילה מישקים ל-VM/SP, SQL/DS, LISP/VM ועוד.

התוכנה מכילה שני סוגים של cut:

- CUT(), המשחרר את המחסנית trail stack, שמכילה מידע בנושא של הצבות (ציינו בספר, כי בדרך כלל אין מבצעים את ההצבה תחת המשתנה, אלא רק מציינים את הקשר).
- /(), אשר פעולתו מהירה מזו של CUT(), אולם המחסנית איננה מתרוקנת ובשל כך, לעיתים יש צורך להיעזר ב-Garbage Collector.

התוכנה מכילה אופרטורים, המכונים "connectors". הם מתארים את הקשרים הלוגיים NOT, NOTNOT, OR, AND, IF-THEN, IF-THEN-ELSE, באופן שונה במקצת מהגדרת פרדיקטי המערכת בעלי שמות דומים שהכרנו במהלך הספר.

תוכנת PROLOG פועלת גם במערכת ההפעלה MVS וקרוייה MVS/PROLOG.

בגירסאות פרולוג של יבמ יש הבדלים קלים בהשוואה לתקן "אדינבורו".

```
garbage collector called by ADDAX
      ls          gs         trail       rules
   388508      81600        20288       32768       22572       19924        2216
   388508      81728        20412       32768         256       44156         300
time 167ms pass 1
```

Garbage Collector Display

This display is interpreted as follows:

Area	free bytes before	free bytes after
local stack (ls)	388508	388508
global stack (gs)	81600	81728
trail stack	20288	20412
sequential free space for rules	32768	32768
logically deleted rules	22572	256
chained part of rules' free space	19924+2216	44156+300

איור VM/PROLOG: מסך של Garbage Collector והסברים

יישומי פרולוג בניתוח מערכות

ראובן גולדשטיין
נמסר באדיבות סיון מחשבים בע"מ

1. מבוא

בחברת "סיון מחשבים" פועלת מחלקת פיתוח בפרולוג, העוסקת בתכנון ופיתוח של כלי עבודה ממוחשבים לניתוח מערכות מידע. כלים אלה נועדו לסייע בשתי משימות עיקריות:
- ליווי מנתח המערכות במהלך פעילויות התכנון השונות הקשורות במחזור החיים של פיתוח וניהול פרוייקטים.
- הקניית שיטות עבודה ואמצעי ביקורת במסגרת הכשרת כח-אדם בענף ניתוח המערכות.

2. יתרונות הפיתוח בפרולוג

בחירת שפת פרולוג כשפת היישום לפיתוח מערכות, נעשתה בעקבות בחינה של היתרונות של השימוש בשפה זו. יתרונות אלה מסווגים לשני ענפים עיקריים: הפתרונות שמציעה שפת פרולוג לבעיות עקרוניות בהנדסת תוכנה, והקשר ההדוק הקיים בין אופי השפה והנושא המיושם.

היתרון המשמעותי ביותר של פיתוח יישומים בשפת פרולוג נובע מאופייה הייחודי של השפה: את פרולוג ניתן לסווג כ-"שפת הגדרות תיאורית" (Descriptive Specification Language). כלומר, קיימת אפשרות מעשית לעבור משלב הגדרת הדרישות ותיאור המערכת אל שלב היישום וההרצה, ללא צורך במעבר דרך שלבי קידוד מייגעים. עובדה זו גורמת לחיסכון בזמן פיתוח יקר, ומאפשרת להציג אב-טיפוס ראשוני של המערכת כבר בשלבים מוקדמים של תהליך הפיתוח. היכולת לבצע מעבר טבעי מההגדרה אל היישום מורידה באופן משמעותי את עלויות התחזוקה והניפוי, שכן יישום המערכת מהווה העתקה ישירה של מפרטי הפרוייקט.

שפת פרולוג מציעה באופן טבעי פתרונות פשוטים לדרישות ובעיות עקרוניות בהנדסת תוכנה. הכתיבה בפרולוג כוללת פירוק היררכי של הפרוייקט ויצירת אוסף של "קופסאות שחורות" לטיפול בתתי-בעיות. קבוצת הגדרות (פרדיקטים) העוסקת במשימת משנה כלשהי, עוברת באופן עצמאי את שלבי הניסוח והבדיקה ומשתלבת בהמשך במערך הפתרון הכללי.

במסגרת היישום בפרולוג נפתרת בעיית התיעוד. הניסוח התיאורי של פרולוג מהווה למעשה גם את התיעוד המיידי של המערכת. בדרך זו, התכנות בפרולוג מייצר כתוצאת לוואי את מסמכי התיעוד הנלווים לפרוייקט.

השיטות והטכניקות השונות הנהוגות בניתוח מערכות מידע ניתנות ברובן להכללה על ידי ציון אופי העבודה של מנתח המערכות. פעילויות אלו כרוכות בפירוק היררכי של תהליכים, מבנים ארגוניים ודרישות, ועוסקות ביחסים בין תהליכים לבין נתונים, בין תכנון לבין ביצוע וכדומה. בעת מיכון תהליכי ניתוח מערכות, יש יתרון רב לשפת פיתוח המאפשרת טיפול במונחים אלה. מיחשוב תהליכי ניתוח מערכות בעזרת שפה בעלת כלים בסיסיים לטיפול ביחסים ובקשרים מהדקת את הקשר בין תהליך פיתוח המערכת לבין הנושא המטופל, ומאפשרת הדמייה מלאה של תהליכי ניתוח לוגיים.

3. יישומים בפרולוג לניתוח מערכות

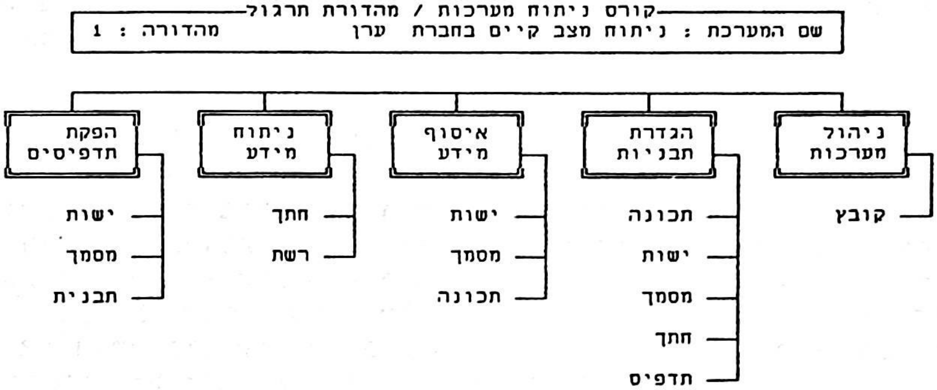
כלי העבודה הממוחשבים לניתוח מערכות מוצעים כתחנת עבודה לניהול מידע. תחנת עבודה זו מהווה כלי פשוט ללימוד ונוח להפעלה, ומיועדת ללוות את מנתח המערכות בכל שלבי מחזור החיים של תכנון ויישום פרוייקטים.

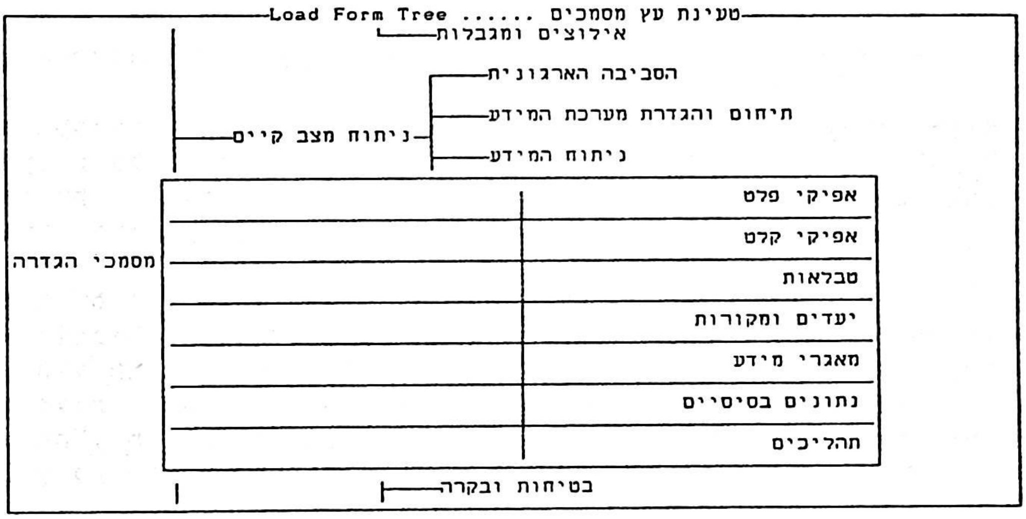
גרעין התוכנה הינו מילון נתונים פתוח למשתמש (Customized Open Ended Data Dictionary) המאפשר לתכנן, לנהל, לתעד ולנתח מידע מכל סוג המעורב בשלבי הניתוח של מערכות מידע. המערכת כוללת אפשרויות להגדרת ועיבוד מידע, שפת שאילתות גרפית יעילה ו"חכמה", מחולל דוחות, רשתות יחסים, עצי מסמכים, וגם מערכת ייעוץ לתכנון מאגרי נתונים.

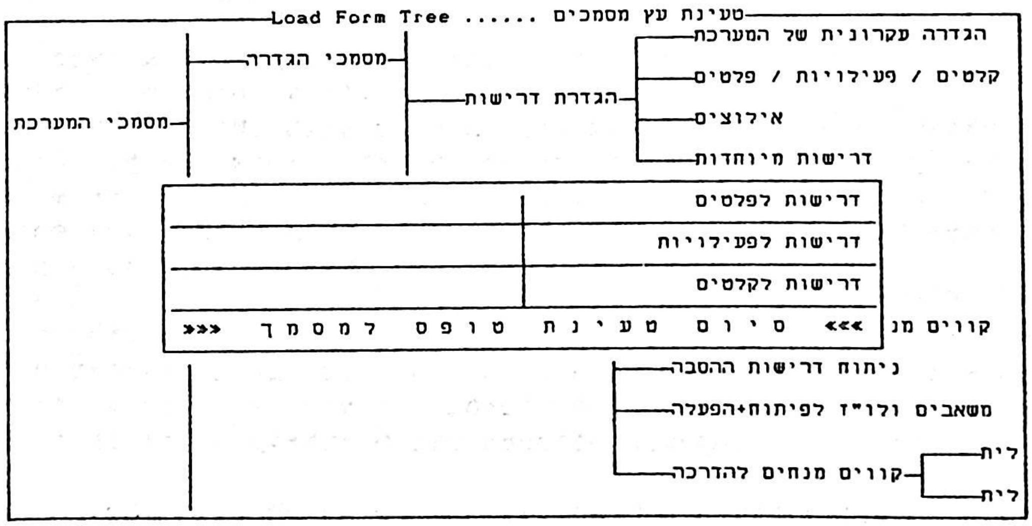
המשתמש יכול להגדיר כל מסמך, מושג ויישות (עצם) המעורבים בפעולות הניתוח. לאחר שלב ההגדרה (Meta Data Requirements) ניתן לבצע איסוף ועיבוד מושגים, כגון תהליכים, אפיקים, תכניות, מודולים, טבלאות וכו'. המערכת מאפשרת להגדיר יחסים בין היישויות השונות, כגון שדות בטבלה, אפיקי קלט לתהליך, מודולים בתכנית ועוד. איסוף המידע מתבצע הן בשיטה הקונבנציונלית (Bottom Up) והן בשיטה המתאימה להגדרות אב-טיפוס (Top Down). המשתמש יכול לשלוף מידע מבוקש בעזרת שפת שאילתות "חכמה" ולהפיק דוחות במבנה ותוכן רצוי.

בנוסף ליתרונות המידיים, המערכת מסייעת להקניית שיטות ותקני עבודה ביחידות מחשב. שלב ההגדרה של התוכנה מאפשר למנהל פרוייקט הפיתוח להחליט על תצורת מסמכי הניהול הנדרשים, ולהנחות את משתתפי הפרוייקט על-ידי שימוש במתכונת שנקבעה.

הערת העורך: הצגה מקיפה של יתרונות השימוש בפרולוג ניתנה בפרקי ההקדמה והמבוא.


קורס ניתוח מערכות / מהדורת תרגול
שם המערכת : ניתוח מצב קיים בחברת ערן
מהדורה : 1
ניהול מערכות
הגדרת תבניות
איסוף מידע
ניתוח מידע
הפקת תדפיסים
קובץ
תכונה
ישות
מסמך
חתך
תדפיס
ישות
מסמך
תכונה
חתך
רשת
ישות
מסמך
תבנית


טעינת עץ מסמכים ...... Load Form Tree
אילוצים ומגבלות
הסביבה הארגונית
תיחום והגדרת מערכת המידע
ניתוח מצב קיים
ניתוח המידע
אפיקי פלט
אפיקי קלט
מסמכי הגדרה
טבלאות
יעדים ומקורות
מאגרי מידע
נתונים בסיסיים
תהליכים
בטיחות ובקרה


טעינת עץ מסמכים ...... Load Form Tree
הגדרה עקרונית של המערכת
מסמכי הגדרה
קלטים / פעילויות / פלטים
הגדרת דרישות
מסמכי המערכת
אילוצים
דרישות מיוחדות
דרישות לפלטים
דרישות לפעילויות
דרישות לקלטים
קווים מנ
««« ס י ו ם   ט ע י נ ת   ט ו פ ס   ל מ ס מ ך »»»
ניתוח דרישות ההסבה
משאבים ולו"ז לפיתוח+הפעלה
לית
קווים מנחים להדרכה
לית

Collect Entity Data ...... איסוף נתונים לישות

שם ישות: תהליך

סוג פעולה: עדכון נתון

שם הנתון (עברי): איתור חריגי נוכחות

אפיק מידע:
- דרישה לדווח חריגים
- דרישות לדווחים
- הודעה על דרגה
- הודעת עובד חדש/חריג
- הוראת בדיקת חריגים
- החלטות דיוני דרוג
- הקצאת חופשה/מחלה
- חוות דעת
- חומר לתיוק
- חריגי נוכחות
- חריגים
- ט. דווח חופש/מחלה/מילואים
- ימי חופשה/מחלה
- כרטיסי נוכחות
- כרטיסים וטפסים

קוד זיהוי	
מטרה	לצורך
אפיקי קלט	הוראת
אפיקי פלט	דרישה
בעיות זמינות	
בעיות דיוק	אין מ
בעיות יעילות	נמוכה

Collect Entity Data ...... איסוף נתונים לישות

שם ישות: אפיק מידע

סוג פעולה: עדכון נתון

שם הנתון (עברי): אשורי עובדים

סוג האפיק:
- מקור
- יעד
- ביניים

שמות נרדפים	טופס
סוג האפיק	מקור
מטרה	אסמכתא לסיבת העדרות @/
כמות	80
תכיפות באפיק	אחת לחודש
אמצעי טכני	מכתב, טופס
תוכן האפיק	תאריך אישור העדרות @/

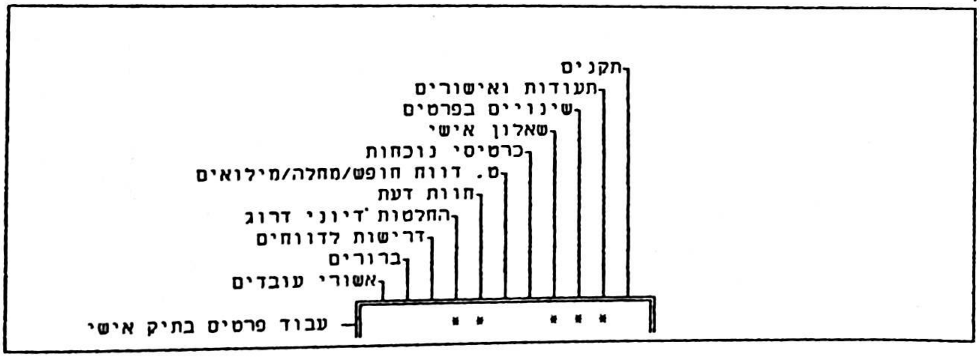
Display Relations Network ...... הצגת רשת יחסים

סוג רשת היחסים: קשרים בין נתוני ישות/חתך

סוג הקשר: אפיקי קלט

קבוצה מכילה (שורות): *תהליך <--- "החלטות דרג"

קבוצה מוכלת (עמודות): *אפיקי קלט (מקור)



פרולוג שפת בינה מלאכותית הינו הספר המקיף הראשון שפורסם בעברית בנושא פרולוג. שפת הפרולוג משלבת בתוכה פשטות של שפה הצהרתית, הפועלת על פי כללים לוגיים ברורים, עם תחכום המאפשר ניתוח ותכנות של תהליכים מורכבים ומסועפים. זוהי שפת **"הדור החמישי"**, המיישמת היום את הטכנולוגיה של העתיד.

"המחברת עשתה עבודה יסודית בריכוז החומר ובהצגתו בצורה מובנית ונוחה ללימוד... ספר לימוד זה ממלא חלל גדול בחומר כתוב עברית בתחום הנושאים של בינה מלאכותית. בכך אין ספק שהוא יתרום להעלאת רמת המודעות והידע בנושאים אלה, ההופכים במהירות לשטחי לימוד ועבודה מרכזיים בנושאי המיחשוב ומערכות המידע"

פרופ' פיליפ עין דור, אוניברסיטת תל אביב (מתוך ההקדמה לספר)

הספר מתאים לדרישות מערכת החינוך ותכניות הלימודים במוסדות אקדמאים ומכללות. הוא מיועד למתחילים בתכנות, להם יהיה נוח מאד להתחיל באמצעות שפת פרולוג את ההיכרות עם עולם המחשב; אנשי מחשב ואקדמיה מנוסים יכירו כלי חשוב בעל עוצמה רבה, שנועד ליישומים מתוחכמים.

המחברת שמה דגש על בניית שלבי הלימוד: לימוד התשתית הלוגית של השפה בעזרת לוגיקה פסוקית, עקרונות של תכנות לוגי, תכנות בשפת פרולוג, והצגה של מספר יישומים מתקדמים. במסגרת זו ניתן הסבר מפורט בנושא **מערכות מומחה:** תאור של מערכת והנחיות לבנייה של קונכיה (shell) של מערכת מומחה.

בספר תמצא מאות דוגמאות קלסיות מן הספרות ודוגמאות הלקוחות מחיי היומיום. בכל פרק ניתנים תרשימים, תרגילים פתורים מלווים בהסברים, הדפסי הרצה במחשב (trace), תרגילים נוספים מיועדים לחזרה ושינון ומהווים מסלה לימודית בעלת תוכן מוגדר.

הספר נכתב על ידי **ד"ר בלה צירולניקוב** אשר סיימה לימודי תואר שלישי בפקולטה למדעים מדוייקים באוניברסיטת תל־אביב. בעלת נסיון רב בהוראה ובהדרכה בתחומי המחשבים באוניברסיטת תל־אביב, בטכניון ובאוניברסיטה הפתוחה.



הוצאת הוד־עמי
לספרי מחשבים בע"מ